May 2020

Power Bounded Computing on Current & Emerging HPC Systems

Pengfei Zou
Clemson University, pzou@g.clemson.edu

Power Bounded Computing on Current & Emerging HPC Systems

A Dissertation
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy
Computer Science

---

by
Pengfei Zou
May 2020

Accepted by:
Dr. Rong Ge, Committee Chair
Dr. Valerie Taylor
Dr. Amy Apon
Dr. Feng Luo
Dr. Xizhou Feng

Abstract

Power has become a critical constraint for the evolution of large scale High Performance Computing (HPC) systems and commercial data centers. This constraint spans almost every level of computing technologies, from IC chips all the way up to data centers due to physical, technical, and economic reasons. To cope with this reality, it is necessary to understand how available or permissible power impacts the design and performance of emergent computer systems. For this reason, we propose power bounded computing and corresponding technologies to optimize performance on HPC systems with limited power budgets.

We have multiple research objectives in this dissertation. They center on the understanding of the interaction between performance, power bounds, and a hierarchical power management strategy. First, we develop heuristics and application aware power allocation methods to improve application performance on a single node. Second, we develop algorithms to coordinate power across nodes and components based on application characteristic and power budget on a cluster. Third, we investigate performance interference induced by hardware and power contentions, and propose a contention aware job scheduling to maximize system throughput under given power budgets for node sharing system. Fourth, we extend to GPU-accelerated systems and workloads and develop an online dynamic performance & power approach to meet both performance requirement and power efficiency.

Power bounded computing improves performance scalability and power efficiency and decreases operation costs of HPC systems and data centers. This dissertation opens up several new ways for research in power bounded computing to address the power challenges in HPC systems. The proposed power and resource management techniques provide new directions and guidelines to green exscale computing and other computing systems.

Author's Publications on this topic

The work in this document is partially based on the following related publications.

1. **P. Zou**, A. Li, K. Barker and R. Ge. Indicator-Directed Dynamic Power Management for Iterative Workloads on GPU-Accelerated Systems. Accepted by CCGRID2020.
2. **P. Zou**, A. Li, K. Barker and R. Ge. Fingerprinting Anomalous Computation with RNN for GPGPU-Based HPC Machines. In *2019 IEEE International Symposium on Workload Characterization (IISWC19)* (Short Paper)
3. **P. Zou**, A. Li, K. Barker and R. Ge. Fingerprinting Anomalous Computation with RNN for GPGPU-Based HPC Machines. In *The International Conference for High Performance Computing, Networking, Storage, and Analysis (SC19)* (Poster, ACM SRC Bronze Medal Award)
4. **P. Zou**, X. Feng, and R. Ge. Understanding and Mitigating Job Contention on Power Limited Clusters. In *14th International Conference on Networking, Architecture, and Storage (NAS)*, Enshi, China, August 2019.
5. **P. Zou**, D. Rodriges, and R. Ge. Maximizing Throughput on Power-Bounded HPC Systems. In *20th International Conference on Cluster Computing (CLUSTER)*, Belfast, UK, 2018. (Poster)
6. **P. Zou**, T. Allen, C. Davis IV, X. Feng, and R. Ge. CLIP: Cluster-Level Intelligent Power Coordination for Power-Bounded Systems. In *19th International Conference on Cluster Computing (CLUSTER)*, Honolulu, HI, 2017, pp. 541-551.
7. R. Ge, **P. Zou**, and X. Feng. Application-Aware Power Coordination on Power Bounded NUMA Multicore Systems. In *46th International Conference on Parallel Processing (ICPP)*, Bristol, 2017. pp. 591-600.
8. R. Ge, X. Feng, Y. He, and **P. Zou**. The Case for Cross-Component Power Coordination on Power Bounded Systems. In *2016 45th International Conference on Parallel Processing (ICPP)*, Philadelphia, PA, 2016, pp. 516-525.

Outline

Abstract . . . . . **i**

List of Tables . . . . . **vi**

List of Figures . . . . . **vii**

1 Introduction **1**
1.1 High Performance Computing and Power Challenge 1
1.2 Power Aware Computing and its Issues 3
1.3 Power Bounded Computing 4
1.4 Research Challenges 4
1.5 Research Objectives and Contributions 6
1.6 Organization of This Dissertation 7

2 Background and Related Work **12**
2.1 Background 12
2.1.1 Enabling Technologies 13
2.1.1.1 Hardware Technologies 13
2.1.1.2 Software Technologies 15
2.2 Related Work 16
2.2.1 Energy Efficient Computing 16
2.2.2 Power Aware Computing 16
2.2.2.1 Power Aware Scheduler 17
2.2.2.2 Power Aware Computing for Node Sharing Systems 18
2.2.3 Power Bounded Computing 19
2.2.4 Power Bounded Heterogeneous Computing 21
2.3 Problem Statement of Power Bounded Computing 22
2.3.1 Problem Statement on Node Level 23
2.3.2 Problem Statement on Cluster Level 24
2.3.3 Problem Statement on Node Sharing Systems 25

3 Cross Component Power Coordination **27**
3.1 Heuristic Power Allocation 29
3.1.1 The Impact of Power Bounds 29
3.1.2 The Scenarios of Power Allocation 33
3.1.3 Linkage Between Scenarios and Critical Power Levels 34
3.1.4 Category-Based Heuristic Power Coordination 35
3.1.5 Experimental Results and Discussion 36
3.2 Application Characteristics Aware Power Allocation 39
3.2.1 Workload Characterization 39
3.2.1.1 Scalability 39

3.2.1.2 Memory Access Intensity . . . 41
3.2.1.3 Critical Component Power Levels . . . 42
3.2.2 Application Aware Power Coordination Algorithm . . . 43
3.2.3 Experimental Results and Discussion . . . 45
3.2.3.1 Application Characteristics of Evaluated Benchmarks . . . 45
3.2.3.2 Evaluation of Application Aware Power Allocation Strategies . . . 47
3.3 Summary . . . 49

4 Cluster Level Intelligent Power Coordination . . . 52
4.1 Configuration Selection on Node Level . . . 53
4.1.1 Scalability Trend . . . 54
4.1.2 Performance Prediction Model . . . 56
4.2 Power Allocation on Cluster Level . . . 57
4.3 System Design . . . 59
4.3.1 CLIP Components . . . 59
4.3.2 Cluster Power-Bounded Scheduling Algorithm . . . 60
4.4 Experimental Results and Discussions . . . 60
4.5 Summary . . . 64

5 Contention Aware Power Bounded Scheduling . . . 65
5.1 Benefits and Challenge of Resource Sharing . . . 66
5.1.1 The Benefits of Resource Sharing . . . 66
5.1.2 The Challenge of Power Bounded Resource Sharing . . . 67
5.2 Methodology . . . 68
5.2.1 Workload Co-run Throughput Prediction without Power Limit . . . 69
5.2.2 Contention vs. Power Coordination . . . 71
5.2.3 Contention under Power Limiting . . . 72
5.3 CAPS Design and Implementation . . . 73
5.4 Evaluation . . . 75
5.4.1 Experimental Setup . . . 75
5.4.2 Performance Models Validation and Accuracy . . . 75
5.4.3 Workload Sharing Recommendations . . . 76
5.4.4 The Performance of the CAPS scheduler . . . 77
5.4.4.1 System Throughput . . . 78
5.4.4.2 Power Efficiency . . . 79
5.4.4.3 Power Utilization and Stability . . . 80
5.4.4.4 Other Metrics . . . 80
5.5 Summary . . . 82

6 Online Dynamic Performance & Power Management . . . 84
6.1 Characteristics of Iterative workloads . . . 86
6.2 ODPP Design and Implementation . . . 89
6.2.1 Iterative Stage Detection . . . 89
6.2.2 Sampling Methodology . . . 89
6.2.3 Period Extraction and Sensitivity under DVFS . . . 91
6.2.4 Configuration Space Exploration through Hybrid Modeling . . . 94
6.2.5 The Overall Autotuning Framework . . . 97
6.3 Experimental Results and Discussion . . . 97
6.3.1 Platform and Applications . . . 97
6.3.2 Accuracy of Speedup Indicator . . . 98
6.3.3 Accuracy of Performance Predictions . . . 100

6.3.4 Accuracy of Average Power Predictions . . . 101
6.3.5 Application Scenarios . . . 101
6.3.5.1 Minimizing Energy Consumption with Performance Constraints . . 102
6.3.5.2 Maximizing Performance Under A Power Bound . . . 102
6.4 Summary . . . 105

7 Conclusions and Future Work . . . 107
7.1 Research Summary . . . 107
7.2 Research Insights . . . 109
7.3 Future Work . . . 110
7.3.1 Power Bounded Computing for Emerging Hardware . . . 110
7.3.2 Power Stable Computing . . . 111
7.3.3 Self-learning Resource Management . . . 113
7.3.4 Trade-off between Power, Performance and Resilience . . . 113

Bibliography . . . 114

Appendices . . . 124
A Appendix Experimental Platform . . . 124
A.1 Intel IvyBridge . . . 125
A.1.1 NVIDIA TESLA K20 . . . 125
A.2 Intel Haswell . . . 125
A.2.1 NVIDIA TITAN XP . . . 125
A.3 AMD EPYC . . . 125
A.3.1 NVIDIA TITAN V . . . 125
B Open Data and Code . . . 126

List of Tables

3.1 Optimal allocation and critical component vs. power budget. 34
3.2 List of benchmarks used in heuristic power coordination 37
3.3 List of benchmarks used in this study 45

4.1 The Haswell hardware events used in sample configurations for prediction. 56
4.2 List of benchmarks used for cluster power coordination 61

5.1 Metrics used in CAPS for workload interference prediction. The metrics are platform specific and may be subject to change on other platforms. 70
5.2 Throughput of job pairing and collocation decision under abundant power ($P_b = P_\infty$) 71
5.3 The list of benchmarks used to evaluate CAPS. 76
5.4 Comparison of used strategies to improve throughput under power bounds 77
5.5 Metrics of compared methods without a power bound. 82
5.6 Metrics of compared methods under power bound of 1760 watts. 82
5.7 Metrics of compared methods under power bound of 1600 watts. 82
5.8 Metrics of compared methods under power bound of 1440 watts. 83

6.1 The list of benchmarks used to evaluate ODPP. 98
6.2 The description of comparison approach. 102

A.1 The parameters of server nodes. 124
A.2 NVIDIA TITAN V, TITAN XP and Tesla K20 configurations [2]. 126

List of Figures

1.1 Performance development of supercomputers since 1992. (captured from `https://www.top500.org/`) . . . 2
1.2 The overview relationship between each research component. . . . 11

2.1 Power allocation for NUMA-based systems. On the NUMA-based systems, application can selectively activate cores and shift power between subcomponents for performance under a power bound. . . . 14

3.1 Approximate distribution of peak power usage by hardware subsystem in a modern data center. The figure assumes two-socket x86 servers and 12 DIMMs per server, and an average utilization of 80%. The figure is copied from Barroso et al. [8]. . . . 28
3.2 The impact of power allocation on application performance and power consumption of the HPCC Star Random Access benchmark. The performance units (UP/s) means number of updates per second. The top and bottom x axes designate the power budgets of processors and memory respectively. . . . 30
3.3 The performance impact of cross-component power allocations on HPCC Star DGEMM (a) and Star Random Access (b) benchmarks. The experiments are evaluated on a node with two 10-core IvyBridge processors. Each MPI process occurs in one core. Table 3.2 listed the detail parameters. . . . 31
3.4 Categorization of power allocation scenarios. The plots of application performance (a) and actual power consumption (b) for different power allocations between processors and memory modules visually reveal six categories of power allocation scenarios. . . 32
3.5 Critical power values for CPU and memory and their relations with RAPL power limiting mechanism. . . . 35
3.6 Performance and power profiles of compute-intensive and memory-intensive applications. While all benchmarks share similar patterns of allocation scenarios, each has application-specific patterns and power ranges for each component for the scenarios. 38
3.7 Comparison between heuristic power coordination and the optimal power allocation identified from experiments. . . . 38
3.8 Application characteristics clustering. The scalability is measured on the two 12-core Haswell sockets node. . . . 40
3.9 The performance and power of SP and FT. This figure shows (1) parallel scalability varies with applications; (2) segmented linear models can be used to approximate the scalability curves and estimate the number of activated cores for a given power bound. 41
3.10 The impact of core affinity on performance and power. This figure shows: (1) the degree of impact differs between the compute intensive application and the memory intensive application; (2) for memory intensive applications, distributing cores to two sockets may result in better performance but a higher power than placing cores on the same socket. . . . 42
3.11 Application aware power coordination strategy. . . . 44

3.12 Parallel speedup and memory bandwidth of different benchmarks under several pivot configurations. The experiment is then evaluated on the Haswell node. 46
3.13 Applications' critical power levels. 47
3.14 Performance comparison of different power allocation methods. 51

4.1 Scalability trends of *linear* (4.1a), *logarithmic* (4.1b), and *parabolic* (4.1c) applications. 55
4.2 Performance impact of processor power budget for *linear* (4.2a), *logarithmic* (4.2b), and *parabolic* (4.2c) applications. 55
4.3 Overview of CLIP 59
4.4 Performance comparison of different power allocation methods under high power budgets 63
4.5 Performance comparison of different power allocation methods under low power budgets 63

5.1 Throughput comparison between course grained and fine grained (collocation) resource scheduling. 66
5.2 The relative system throughput when collocating EP & STREAM under different power budgets and (CPU, MEM) power coordinations. 68
5.3 The performance and power estimation models. 70
5.4 Impacts of power coordination between CPU and DRAM on workload throughput. . 72
5.5 STP vs. power budgeting. Throughput decreases as the total power budget P_b decreases. Complementary jobs may become contentious when the power budget is inadequate. 72
5.6 The two level power coordination framework for job collocation. 75
5.7 System throughput comparison. The figure shows the execution time for all the jobs under different power budgets. Less time indicates better performance. 78
5.8 Power efficiency comparison. This figure shows the Energy-Delay products. Smaller values mean higher efficiency. 79
5.9 System power traces. The figure shows that CAPS maintains a high power utilization at a constant level. 80
5.10 Nodal power traces. CAPS reduces the imbalance of power distribution between nodes comparing with the other two power bounded methods (POW, and Twins). 81

6.1 A 10s screen-shot of the GPU power, core utilization, memory utilization trace for training *Inception3*. The trace is obtained on NVIDIA Titan-XP while running Inception3 CNN model with ImageNet dataset and 64 images per batch. 87
6.2 *CoMD*'s phase change and corresponding power during two iterations. The trace is obtained on NVIDIA Titan-XP while running CoMD. 88
6.3 The fully trace of running *CoMD*. 90
6.4 Loss rate of `nvidia-smi` sampling under different benchmarks and user specified sampling interval. 90
6.5 The amplitude after transferring the 10s resource utilization trace of Inception3. . . 92
6.6 The resource utilization trace's FFT Max frequency shifts left while decreasing GPU Graphic frequency. 92
6.7 The changes of actual performance and period extracted from resource utilization trace with *Inception3*. The base is at the nominal or the highest stable frequency. . . 93
6.8 Application normalized speedup and energy consumption under different core frequencies. 94
6.9 The overview of proposed framework. 96
6.10 The error of transform extracted period to normalized speedup. 99
6.11 Prediction error of speedup. The AM model adopts measured or extracted speedup values as initial point to estimate other configurations with piecewise interpolation. . 100
6.12 Prediction error of power consumption. 101

6.13 Comparison of application minimize energy error under different performance requirements. 103
6.14 Comparison of application performance under different power bounds. 104

7.1 Per-Switchboard power data for each of Cielo's five switchboards. [80] 112
A.1 The architecture of the cluster for evaluation in the proposal. 124
A.2 The architecture layout of NVIDIA K20 GPU. 126
A.3 The architecture layout of NVIDIA TITAN XP GPU. 127
A.4 The architecture layout of NVIDIA TITAN V GPU. 128

Chapter 1

Introduction

1.1 High Performance Computing and Power Challenge

High Performance Computing (HPC) has played and will continue to play a fundamental role for supporting scientific discovery and economic growth. Both of science and engineering disciplines heavily rely on extensive use of mathematical models to analyze observations and make predictions. Due to the mathematics and large amount of data needed to represent the problem, obtaining and executing the models is very computationally intense. The HPC platform provides the massive computational power for scientists and engineers to tackle such problems fast.

Due to the development mathematical modes and increasing amounts of data, the demand for higher performance capability increases very fast. As shown in Figure 1.1, the performance of the top 1 supercomputer has been consistently increasing from 10^9 Flop/s to 10^{18} Flop/s. From Argonne National Laboratory's report [6], the first exascale supercomputer will be delivered in early 2021. The future El Captain [63] will outperform the total computational capability of today's top 500 supercomputers.

While the top supercomputers' performance keeps increasing fast, the supercomputers also consume much more power and bring power management problems. As of November 2019, the world's fastest supercomputer – Summit, consumes 13 Megawatts (MW) of power. Meanwhile, the world's largest data centers consume 100-150 MW of power. The super high power consumption not only costs a large amount of energy, but also brings power supply, power management and cooling challenges. The scalability of the next generation HPC systems is increasingly constrained

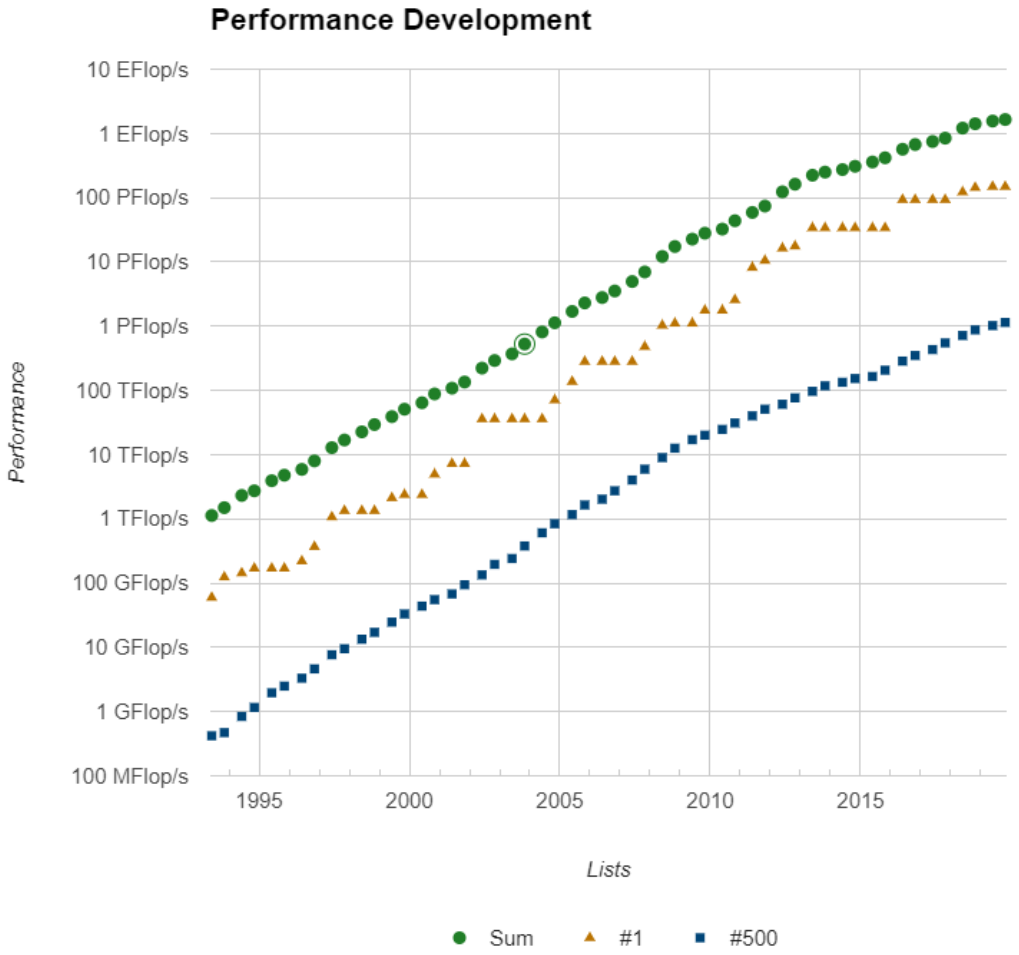


Figure 1.1: Performance development of supercomputers since 1992. (captured from `https://www.top500.org/`)

by the power requirement. In fact, power has become one of the most important concerns for the next generation of supercomputers and commercial data centers. Due to the physical hardware limitation and operating cost, the power efficiency has to increase significantly to meet the increasing demands. The U.S. Department of Energy (DOE) has set the target to improve power efficiency consistently on the way towards exascale computing [64, 71]. The Exascale Computing Project (ECP) has set the goal to improve power efficiency by 20% in a power constrained environment [83].

1.2 Power Aware Computing and its Issues

Sustaining the performance growth under a power limit triggered research interests in both semiconductor industry and system management community. The former is focusing on developing novel and more efficient hardware, and the latter is focusing on improving the utilization of the available hardware. Both directions have changed the way to think about the role of power in computing and are complementary to achieving exascale computing.

Hardware evolution focuses on designing efficient processor, memory, storage, and network hardware device. Many-core processors and GPU accelerators have been introduced as major computing components for high throughput applications. As of November 2019, 5 of the top 10 supercomputers are equipped with powerful GPU accelerators [3]. The translation to GPU clusters requires optimizing application performance on GPU accelerators. Hardware also integrates performance state and adjustable ability for system or application to dynamically trade off between performance and power. Emerging architectures introduce a tradeoff between significant power efficiency and difficulties in system management due to heterogeneity.

The system management enables the available system hardware to operate more efficiently and increase application performance, system throughput or energy efficiency. System management technologies aim to maximize available hardware capability and allocate resources among different parts in system to achieve the performance and energy goals. System management technologies not only target the efficiency goals for future exascale systems but also can improve current system performance and energy efficiency.

Traditionally, from system level, researchers have predominantly investigated power-aware computing to improve energy efficiency or save power consumption for HPC and data centers. Power aware computing is a system design approach that explores power-aware technologies to facilitate optimal performance with as low power usage as possible [88, 84, 18, 36, 104, 34, 81, 89, 28, 60, 26, 67, 23, 53, 45, 82]. A majority of power aware researches focus on power reduction via effectively coordinating power aware components including processors, memory, disk, network and their combination. Researchers also utilize software throttling such as controlling software execution by adjusting the degree of concurrency and the number of participating cores.

Power aware computing is able to save energy consumption or improve energy efficiency, but fails to address the power challenge *and* the demanded performance and scalability on future

exascale HPC systems. Power aware computing does not take power as a key limited resource for allocation and coordination. As a result, power aware computing is not able to maximally transfer the available power budget as application performance or system throughput.

1.3 Power Bounded Computing

In this dissertation, we propose power bounded computing to address the power challenge and meet the demand for scalable performance on current and emerging high performance computing systems. Power bounded computing considers power as a scarce resource and aims to maximally exploit available power budget to increase application performance and system throughput. It overprovisions hardware components but dynamically manages them to ensure the system operating within the given power bound. Power bounded computing tackles the problem from the system side and exploits the features and takes the advantages of hardware techniques.

Both power bounded computing and power-aware computing rely on the availability of power-aware components, which support a set of power-performance states and can transition from one state to another as instructed by a user or a program. Both approaches seek to adapt the system's state to match the workload. However, power bounded computing is different from power-aware or power capping technology in its objectives and employed techniques. For power bounded computing, given a computer system with performance and power adaptable components, the system needs to identify how to configure the hardware resource and power allocation to achieve the performance and energy goals. Unlike power-aware computing that assumes power is abundant, power-bounded computing views that power is scarce and must be scheduled to maximize performance. It identifies the right power budget for applications under study and allocates the remaining available power to other applications. As a result, power bounded computing ensures that power is maximally translated to performance and is within the given power budget, while power aware computing may fail to achieve the best performance or meet the power budget.

1.4 Research Challenges

Implementing power bounded computing is quite difficult. Normally, improving application or system performance conflicts with reducing power consumption. To improve or maintain the

application performance in a power constrained system, the system has to determine, allocate and manage available resources with a high precision. Here, we list multiple challenges for optimizing application performance in a power-constrained system:

I Power bound guarantee for a system. In the future HPC system, the power supply infrastructure and cooling service may not be able to support all system components to operate in the maximal performance levels. To ensure safe system operation, there must be a cap for the power distribution in all levels and system shall maintain the application and system performance. At the same time, the system needs to minimize the impact of such power capping at all levels.

II HPC workloads variance. HPC systems run a large amount of different workloads in different scientific disciplines, and the workload characteristics varies significantly among each other. Even for a single workload, different problem sizes or input parameters could change the workload characteristics. Broadly they differ in terms of memory or computational requirements; and could be power efficient or greedy. HPC workloads also have different levels of scalability. A simple power allocation strategy may work well for workloads with specific characteristics, but fails for others. The system level power management technologies need to be adaptive for workloads with all kinds of characteristics.

III Power distribution in a power bounded system. System components may be deactivated to maintain power constraints at all levels. The power allocation at one level may also impact the distribution in another level. As there are a lot of configurations in each level, it is impossible to exhaustively explore all configurations to figure out an optimized configuration for workload execution in a power bounded system. The system level management has to investigate deeply on the impact of power cap on different components and figure out an optimal configuration or close approximation.

IV Job scheduling in a power bounded system. Usually, multiple jobs are concurrently running on a system. Different jobs may require different amounts of resource, and generate different levels of power consumption. For a lot of HPC systems and commercial data centers, different jobs also share nodes and components between each other. The jobs will compete for shared resources (e.g. memory bandwidth, network bandwidth, processor cache, etc). In a power

bounded system, power constrains the amount of activated hardware resources and their capacity for workload execution, and thus induces or aggravates contention, particularly on the memory hierarchy. Therefore, the system has to consider job allocation for impact of both power bound and contention on shared hardware resources.

V Performance requirement in a power bounded system. While the system imposes a power constraint, the system shall keep meeting the quality of service (QoS) while maximizing the power efficiency. Therefore, the system needs to estimate accurately about the impact of the resource configuration on the workload in a power bounded system online. Without prior knowledge or profiles of the workloads, the system has to estimate different configurations with the features obtained online. To monitor the application performance online, internal monitoring API is traditionally integrated with workloads, requiring application developers' effort and intrusive code instrumentation. To enable online monitoring of application performance, the system needs to identify features that is related to workload performance and build the transformation between workload performance and features.

1.5 Research Objectives and Contributions

This dissertation aims to maximize application and system performance for power bounded high performance computing systems. The objective of the dissertation is to develop innovative methods to enable power bounded computing at both node and cluster levels, for both homogeneous and heterogeneous systems. The overall goal of the dissertation is to develop resource and job scheduling systems which aim to resolve the conflicting needs of scaling performance and limiting power consumption. The goal is aligned with the DOE critical missions and aims to contribute for solving the power challenge problem for exascale system. The following summarizes the research objectives and contributions of this dissertation.

- develop power bounded computing frameworks. The power bounded computing framework should consider the power consumption and performance as first priority to address the challenge I. The frameworks always ensure that the system power consumption will not surpass the limited power budget. The frameworks involve analytical and practical technologies to meet the requirements of (1) maximizing performance within a power bound at cluster, node or component level, and (2) adapting to diverse demands of applications and systems.

- evaluate the proposed power bounded computing with multiple workloads across different benchmark suits to address the challenge II. The evaluation workloads should have generalized diversity to represent a variety of HPC workloads. In the dissertation, we collect evaluated workloads from multiple benchmark suits. The workloads include compute intensive workloads, memory intensive workloads, highly scalable workloads, poorly scalable workloads, power greedy workloads, power insensitive workloads, etc. These workloads prove that the proposed work are able to address the challenge II – workload variance. These workloads come from typical HPC application kernel, HPC workloads, data analysis workloads and emerging workloads, and are able to represent the workload diversity for HPC systems.

- develop advanced technologies for both homogeneous and heterogeneous power bounded computing systems. We develop multiple components to address the challenge III∼V. The key components include *cross component power coordination*, *cluster level intelligent power coordination*, *resource coordination for node sharing systems*, and *online power management*. Application aware cross component power coordination identifies the critical power levels for core computer components according to application characteristics and available power budget, and improves performance significantly under multiple power bounds. We study its benefits and generalize it to cluster level, and present CLIP (Cluster Level Intelligent Power management) to coordinate power on both node level and cluster level. We investigate the impact of power bound on node sharing system, and present power, hardware and workloads co-scheduling for node-sharing system. Finally, we exploit the periodical pattern presented on GPU resource utilization trace for iterative workloads, and propose ODPP (Online Dynamic Power-Performance Management) for performance requirement and GPU energy efficiency.

1.6 Organization of This Dissertation

This dissertation includes three parts. The first part consists of Chapter 1 and 2, which present statement and background of power bounded computing. The second part includes Chapters 3, 4, 5, 6 in which we describe the details of key technology for power bounded computing. Finally, in Chapter 7, we summarize the results, conclusions and contributions from this dissertation and discuss future research. Here we overview each component as follows:

1. **Cross Component Power Coordination**

High performance power bounded computing is built upon the premise that every compute node in the system can and will operate under a given power budget. Node level power bounding is the foundational stone of large scale system power bounding. By bounding per-node power consumption, a large scale system is capable of reconfiguring itself according to its current workload to achieve better performance under the same power budget.

The nodal power is the sum of the power consumed by all components on a node. Enforcing a nodal power limit requires allocating an appropriate amount of power to each component in a coordinated manner. Cross component coordination is crucial for a compute node to deliver maximum application performance corresponding to a specific power budget, particularly when power becomes such a scarce resource that cannot concurrently meet the maximum demands of all components.

In this dissertation, we focus on power coordination between processor and memory module, as processor and memory modules are the major consumer of dynamic power. We study the impact of different power coordination between the two major components and discover that (1) different applications share categorical patterns with regard to how power allocations among individual components impact application performance and actual power; (2) the per-node power budget must exceed a certain threshold in order to achieve desirable performance and efficiency; (3) there exist workload-specific optimal power allocations under a given power budget and such optimal power coordination can be pinpointed using the heuristics derived from the categorical patterns and a light-wight power-performance profiling.

Cross-component power coordination exploits the patterns of the power bound impact on application performance, and allocate power according to application characteristics and available power budget. Evaluation results shows that coordination power between components according to application characteristics is able to obtain optimal/sub-optimal performance under a power bound [42, 44].

2. **Cluster Level Intelligent Power Coordination**

Cluster level intelligent power coordination scales up the cross component power coordination on a cluster level. Optimally managing cluster level power for HPC workloads requires an intelligent strategy to control the number of participating nodes and allocate the available power budget to different subcomponents (CPU-core, CPU-uncore and memory) within nodes.

Inappropriately assigning nodes can either cause inefficient utilization of the available power or lead to subsystems running at ineffective power levels, thereby delivering inferior performance. For example, if fewer nodes are assigned, each node gets excessive power budget than required and applications' parallelism can't be fully exploited at the cluster level. Contrarily, if more nodes are assigned, each node receives insufficient power to operate the constituent components at their optimal states, leading to significant performance degradation. System power management is challenging and requires careful balance between the cluster, node, and component levels to avoid power waste and performance degradation.

Cluster Level Intelligent Power (CLIP) coordination addresses power bounded computing on multicore-based clusters. CLIP employs application-aware power bounded scheduling for parallel applications on clusters built of NUMA multicore nodes. It characterizes the scalability of parallel applications and their power demands, and accordingly recommends the sub-optimal application execution configuration and power distribution. The framework implementation is hierarchical and consists of two levels: the cluster level determines the number of nodes and the power budget for each node; the node level selectively activates the CPU cores and distributes the available power budget to the CPU and memory within nodes. The framework uses light-weight offline profiling for application characterization, and classifies workloads into three categories. Our evaluation shows that CLIP outperforms prior work significantly for complex applications. The average improvements are close to 20% under low power budgets.

3. **Contention Aware Power Bounded Scheduling for Node Sharing Systems**

   To extend power bounded computing for node sharing system, we propose Contention Aware Power Bounded Scheduling (CAPS) to address the power challenge for job co-scheduling systems. In node sharing system, collocated jobs contend shared resource like memory bandwidth, last level cache, and interconnection. Such contention slows down individual application executions, potentially to an extent that results in worse system performance than executing the jobs in sequence.

   The power limitation further deteriorates the contention between co-located jobs. Power constrains the amount of activated hardware resources and their capacity for workload execution, and thus induces or aggravates contention, particularly on the memory hierarchy. Furthermore, when the total power is limited, balancing power among nodes and components is critical;

under-provisioning on some components can lead to severe contention and performance degradation. Understanding the impact of power allocation at the system, node, and component levels is necessary to mitigate contention and performance interference.

We study how power limiting affects contention between collocated scientific parallel jobs in multicore based clusters, and research effective strategies to mitigate contention and maximize system performance under given power budgets. We present *CAPS*, a Contention-Aware Power-bounded Scheduling that mitigates contention and coordinates power between nodes and components. Overall, *CAPS* embraces two key ideas: (1) infer the contention using applications' performance and power profiles and their variation with power limits, (2) exploit job collocation *and* supportive power distribution across nodes and components to mitigate contention caused by power limits. Depending on the power limits, Results demonstrate that CAPS improves system throughput by 10% or more than job collocations that are oblivious to power limits, and 25% or more than the typical first-come-first-serve scheduling deployed on clusters.

4. **Online Dynamical Power Coordination** Modern high-performance and warehouse computing centers show strong interest in minimizing system power consumption while satisfying customer requested quality of service (QoS). Automating the process online is challenging due to the great complexity — today's hardware components (e.g., CPUs, GPUs, memory, network, etc.) can be configured in several or dozens of frequency/voltage states for satisfying divergent system demands. Given their combination and the emergence of heterogeneity, searching the optimal configuration in the design space online can be timing consuming. Existing work relies on detailed knowledge of workloads, either obtained from extensive offline profiling or intrusive code transformation for fine-grained profiling, which are unable to transparently support unknown workloads.

   We focus on applications exhibiting an interesting feature – iterative or periodic, which is common among conventional HPC and emerging machine learning workloads. We propose an online dynamic power-performance (ODPP) management framework. ODPP runs on the host and dynamically adjusts GPU DVFS configurations to meet performance and power objectives without any code annotation or intrusion. Particularly, ODPP extracts the performance and power indicators for applications from easily obtained GPU resource utilization profiles in

a short episode. The extraction method uses Fourier Transform and incurs negligible cost. ODPP further automatically constructs an accurate model that infers from the indicators how the application's performance and power vary with GPU core and memory frequencies. Aided with the model, for both seen and unseen applications, ODPP can quickly determine the most appropriate DVFS configuration for application execution. Evaluation results on an NVIDIA GPU show that ODPP can improve energy efficiency by over 30% under different QoS and improve performance by more than 8% under different power bounds.

In the dissertation, the components are tightly coupled and together systematically support power-bounded computing at scale. Cross-component power coordination is the base building block and provides support for other components. CLIP further extends cross-component power coordination to the cluster level, and enables power bounded computing for a single job running on multiple nodes. CAPS improves CLIP by considering job contentions on node sharing systems, and allocates hardware resource, power, jobs collaboratively for maximizing system throughput. Finally, ODPP applies power bounded computing on heterogeneous architecture, and coordinates hardware configuration for energy efficiency and performance requirement. Figure 1.2 shows the relationship between each component. The four components together are able to address power bounded computing for large scale systems with different architecture and purposes.

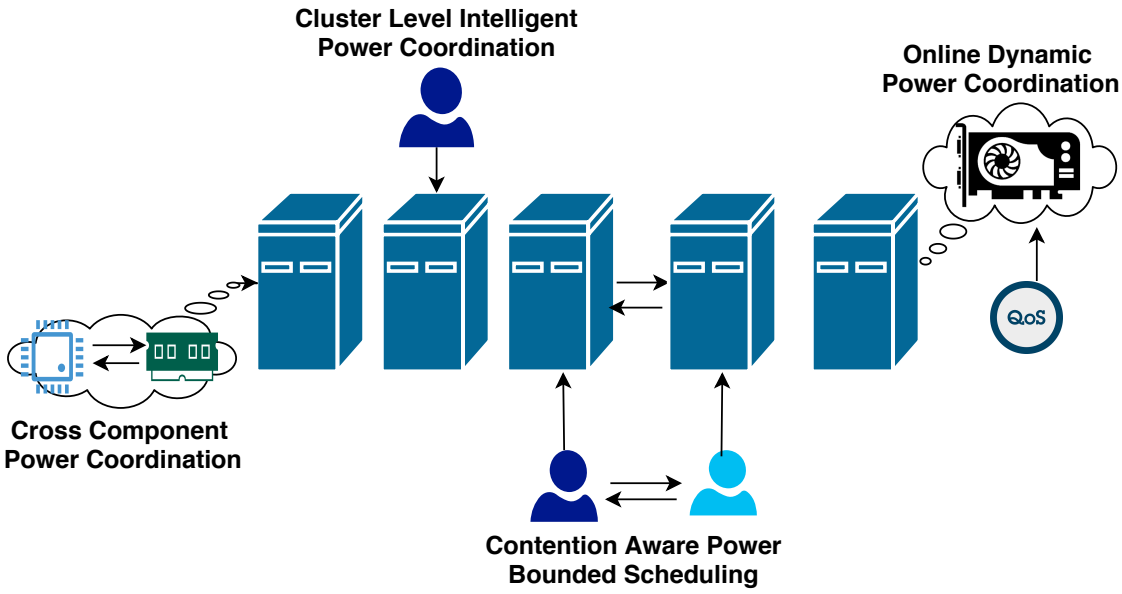


Figure 1.2: The overview relationship between each research component.

Chapter 2

Background and Related Work

In this chapter, we present the background about power bounded computing, existing work related to the dissertation and discuss the theory for power bounded computing. We first introduce the background of the dissertation and overview the state-of-the-art. We broadly classify the related work into four categories: (1) energy efficient computing, (2) power aware computing, (3) power bounded computing, and (4) power bounded heterogeneous computing.

2.1 Background

Over the past decade, HPC system researchers have transformed the goal of only improving HPC system performance to improving both performance and energy efficiency. As shown in Figure 1.1, the top 1 supercomputer's performance increased from about 4 PFlop/s to 100 PFlop/s. At the same time, the power consumption increased from 4 MW to 15 MW. Even though the power efficiency has increased 6 times over the last 10 years, the power efficiency is still far from good enough to support exascale systems. Here we discussed the enabling technologies that support the power efficiency increment.

2.1.1 Enabling Technologies

2.1.1.1 Hardware Technologies

Multicore/Manycore CPU processor: As the end of Dennard Scaling, the core frequency keeps stagnant to avoid power increasing rapidly on processor. To increase the processor performance, the chip manufacturer integrates more cores on each processor. In the past 10 years, on the top 1 supercomputer system, the cores integrated on a processor per node have increased from 6 to 48. The processors have evolved from multicore era to manycore era.

The increment cores on a processor requires workload and system to improve scalability to better exploit the processor parallelism. In many core eras, a lot of workloads may not be able to achieve full scalability due to poor algorithm design. Therefore, the system scalability and power budget have a higher chance to be wasted. The system needs to develop a novel scheduling policy to address such concerns and increase system throughput.

Non-Uniform Memory Access (NUMA) NUMA is a method of configuring a cluster of microprocessors in order to share memory locally. NUMA improves performance and enables system to be expanded easily. Most of today's CPUs adopt NUMA-based architecture to improve system performance and energy efficiency.

On a NUMA-based system, the power coordination is much more complex. Figure 2.1 shows the power allocation on a NUMA-based multicore node. Power allocation on a NUMA system needs to consider both the resource allocation and power allocation between the NUMA connected components. Therefore, the system shall consider the running workload affinity and assign optimal number of threads on each professor to exploit local bandwidth and remote bandwidth well.

Heterogeneous memory The performance gap between processor and memory is larger as the memory speed increases much slower compared with processor. To minimize the performance gap between processor and memory, the system integrates multiple level memory to trade-off between memory bandwidth and memory capacity. For example, Intel Knights Landing processors replace traditional L3 cache as high bandwidth memory (HBM) to decrease the speed gap between L2 cache and memory. On the latest GPU architectures (e.g. NVIDIA Volta V100) utilizes the HBM as major memory components instead of GDDR5 to increase GPU memory bandwidth.

The heterogeneous memory architecture increases the system complexity. The heteroge-

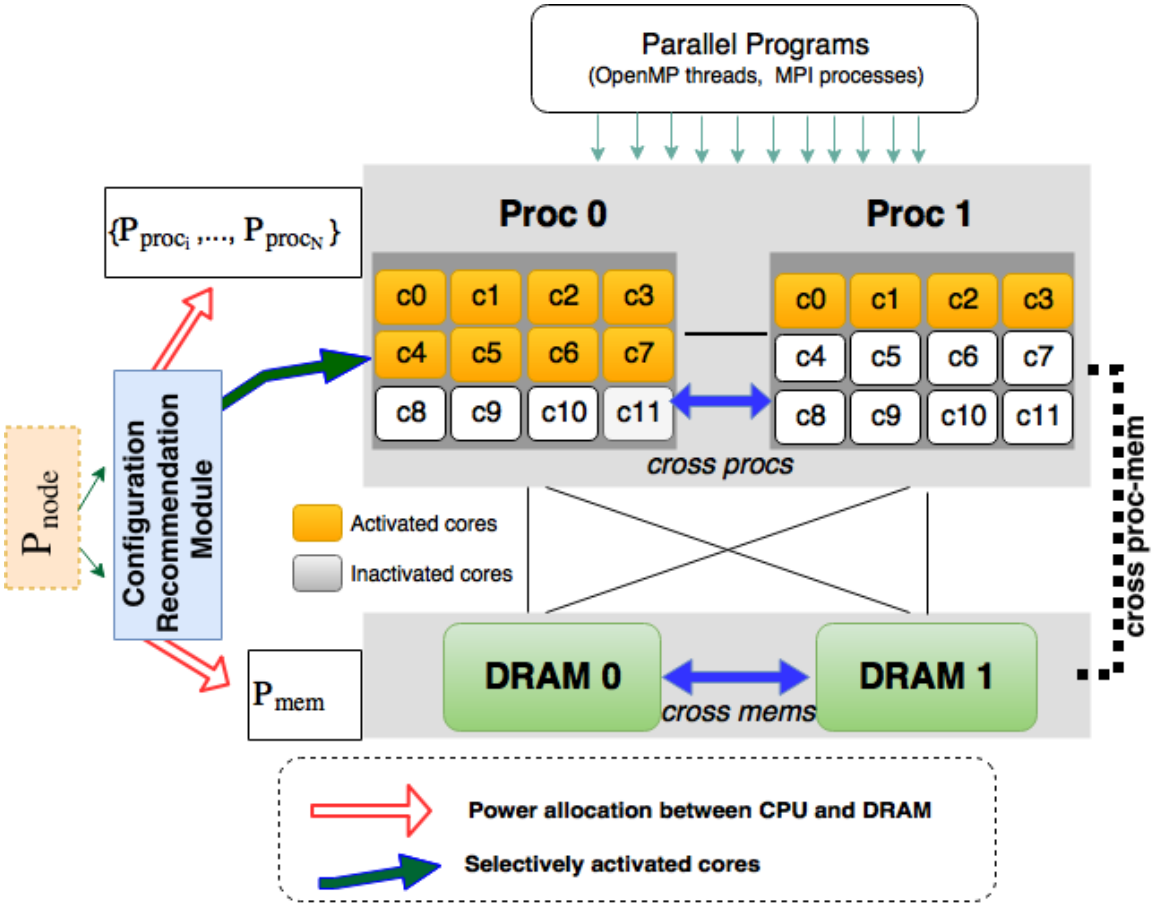


Figure 2.1: Power allocation for NUMA-based systems. On the NUMA-based systems, application can selectively activate cores and shift power between subcomponents for performance under a power bound.

neous memory increases system energy efficiency and proposes new challenges for power bounded computing. To increase application performance under a power limit, power bounded computing shall also consider the capability of workload exploitation on memory hierarchy and allocate power correspondingly on each component.

GPU The GPU accelerators have widely been integrated on HPC systems due to the high performance power ratio of GPU accelerators. As of November 2019, GPUs appear on five of today's top ten supercomputers [3]. These GPU-accelerated systems support not only traditional HPC applications, but also emerging deep learning applications in various disciplines including science, image processing, and natural language processing. The GPU card are also the major power consumers in HPC systems [5]. For example, GPU consumes more than 85% power consumption of all the

computational components on the top 1 supercomputers – Summit [38]. Power bounded computing shall also take GPU accelerators into consideration.

2.1.1.2 Software Technologies

Workload Execution Management One common feature of parallel applications is that most applications can have multiple runtime configurations including number of MPI processes, number of threads, and process/thread placement [46, 23]. Even on the same computing platform with the same workload, changing runtime configuration has major impacts on the system performance and power consumption. The increasing number of cores on a processor improves the complexity of workload execution management.

Core and Memory Affinity In multicore/manycore computing, both data locality and memory contention affect parallel application performance and consequently system power [24, 20]. Binding parallel threads to appropriate processor cores and memory modules has the potential to improve cache locality and application performance. Hardware affinity can be implemented through kernel calls like `sched_setaffinity`, system tools like `taskset` and `hwloc`, runtime environments like OpenMP and OpenMPI, or job schedulers.

Power Budgeting and Power Capping The capability of enforcing an allocated power budget is crucial to power-bounded computing. A power bounded system can safely schedule the available power budgets to those critical computing units only when all units comply with their own power budget. Power capping has been implemented in software, hardware, and a combination of both. Existing software-based power budgeting approaches include dynamic voltage and frequency scheduling (DVFS) [57, 26], joint power management between processor and memory [19, 49], software concurrency throttling [92, 23], and thread packing [23]. Meanwhile, the latest major computer components like CPU, memory, and GPU supports hardware-based power-capping capability [25, 26, 77, 55].

2.2 Related Work

2.2.1 Energy Efficient Computing

Increasing energy efficiency is one of the main goals for both HPC community and commercial data centers. In some HPC systems and commercial data centers, the operating cost is charged by the actual energy consumption. As a result, the energy bill is a major part of the data center operating cost. Researchers focused on adapting the performance state of hardware to save energy consumption within acceptable performance penalty or meet the quality of service (QoS). Woo and Lee investigate and extend Amdahl's law to consider power and energy consumption and suggest many-core runtime should enable dynamical per-core power profiling [100]. Practically, dynamic voltage and frequency scaling enables the system to achieve tradeoffs between energy and performance and is widely used in both HPC and commercial data centers. CPUMiser [41] and Jitter [61] propose different DVFS-based algorithms to sacrifice some performance for energy efficiency. Wang et al. [99] and Li et al. [60] study analytical models with DVFS to predict performance and energy consumption. Adagio [89] implements a runtime system with the goal to maximizing energy saving without impacting performance by using DVFS. In cloud computing environment, DVFS and switching power of nodes on/off are the major techniques for energy saving. Multiple task consolidation heuristics have been presented by researchers [58, 9].

Energy efficient computing is targeted at saving energy consumption and reducing carbon emissions of data centers. The objectives of energy efficient computing and power bounded computing are significant different from each other. Power bounded computing enforce the power bounds on the system and transfer available power budget to system performance maximally. Power bounded computing also increases system energy efficiency by increasing the system performance. Energy efficient computing may fail to enforce the system power bound or deliver best performance under a power budget.

2.2.2 Power Aware Computing

Power aware computing aims to meet the actual power demand of workloads and saving power consumption with little or low impact to workload performance. Power aware computing includes power saving and power capping techniques. Researchers have studied effective solutions to coordinate the hardware components to decrease power or energy consumption. Dent et al. [29]

propose CoScale to manage CPU and memory subsystem energy. CoScale explores the possible frequency of CPU and memory to minimize the energy consumption within the performance bound. Li et al. [62] also develop algorithms to jointly manage processor and memory for energy consumption.

Power capping techniques ensure that the components or systems operate under a power budget, in order to ensure that the system operating at a safe temperature or meet the requirement of power supply infrastructure. There are two ways to implement power limit: hardware-based power capping and software control of workload execution. RAPL [25] is one of the most widely studied technologies to cap the power consumption by hardware. Researches [42, 44, 106, 108, 26, 67] utilize RAPL to limit the system power consumption. Software control approaches utilize algorithms or strategies to dynamically adjust the state of hardware or the level of concurrency. Nornir derives models to estimate performance and power consumption under different configurations and select configuration according to user performance or power requirement [28]. Li et al. [60] study the power-performance characterization and propose a heuristic approach to achieve trade-off between power and performance for shared-memory parallel applications. In Li et al. [60]'s work, they adapt the configuration in two-dimensional space: (1) the possible number of active processors (2) different voltage-frequency levels. PEPON [92] distributes power in two levels for NoC based multicores. On the first level, power is distributing among various types of on-chip resources; on the second level, power is distributed to individual instance of each type of resource. Pack & Cap [23] combines DVFS and thread packing to achieve performance under power caps. Pack & Cap [23] packs threads onto a variable number of cores and enable idle cores enter low-power sleep states to maximize performance under a power budget. Several works proposed combination of software and hardware methods to improve energy efficiency or meet the power budget. Zhang and Hoffmann propose a hybrid software/hardware power capping system named as PUPiL [104]. PUPiL first applies RAPL to implement the power capping on the system and reply on binomial search to identify the optimal configuration that delivers best performance. The configuration space includes DVFS settings, hyper-threading control, TurboBoost control and threads concurrency control.

2.2.2.1 Power Aware Scheduler

Traditionally, First-Come First-Serve (FCFS) is a simple and widely used scheduling policy. Even though FCFS is easy to implement and ensure fairness, but it can block small jobs and leads to fragmentation. As a result, FCFS increases the waiting time and fails to utilize available resource

efficiently. Backfilling algorithms are introduced to deal with the underutilized fragmentation caused by FCFS. Backfilling moves smaller jobs forward to improve resource utilization. The smaller jobs can be allocated to the idle nodes that are left by larger jobs. Backfilling may de-emphasize the job priorities and user fairness in some cases. Observation results reveal that enable backfilling with FCFS could improve system utilization by about 20% for large HPC systems [53]. However, such scheduling policies fail to consider power consumption while allocating jobs and could waste the available system power budget.

As the power become much more important HPC systems, researchers start to integrate power aware algorithms with the traditional job scheduler to improve system power utilization and minimize the impact of throughput by a power bound. In the work [82], Patki et al. present the power-aware backfilling scheduler: Resource MAnager for Power (RMAP). In RMAP, jobs are considered to be moldable and have predicable performance slowdown under a power cap with different configurations. RMAP schedules the jobs by power-aware backfilling and maintain the system power budget. Simulations results demonstrate RMAP can reduce the job turnaround time significantly.

PTune [45] takes the performance variability under a power bound into consideration and proposes a 2-level hierarchical power managing approach. At the macro level, the approach shifts power between jobs to ensure the job get relatively reasonable power budget. At the micro level, the approach tune power node by node to minimize the impact of performance variability. Research in [85] also tackles the performance variability.

2.2.2.2 Power Aware Computing for Node Sharing Systems

Major resource managers like SLURM and PBS initially consider node as the basic resource allocation unit, and upgrade to support a more fine-grained, per-core based resource scheduling. The rationale is that modern nodes comprise dozens of processor cores and running multiple jobs on the same node can effectively increase system utilization and throughput. The challenge is to limit the impact of performance interference for node sharing systems between jobs.

Recognizing the importance of accurately predicting performance interference for node sharing, researchers have investigated various performance models. *DI* and *DIO* schedule threads to evenly distribute miss rate among cache and reduce the effect of contention [10]. *ASM* uses cache access rates to estimate the slowdown of applications due to resources (i.e. caches and main mem-

ory) sharing [97]. Dwyer et al. [32] adopt decision trees to analyze hardware events and develop a model for estimating performance degradation. Dwyer et al. [32]'s study targets at older generations of multicore systems and sequential workloads. Breitbart et al. [11] experimentally show that energy saving and performance improvement can be achieved with co-scheduling a memory bound application and a compute bound application in HPC environment. Sasaki et al. [90] studied the contentions for multiprogrammed workloads on manycore nodes under power capping. In this dissertation, we are targeting at node-sharing HPC systems and further investigate how power limits affect workload contention at the system, node, and component levels.

2.2.3 Power Bounded Computing

HPC system have distinctive characteristics of power consumption and special requirements for power supply. First, HPC system requires high reliability power supply. Therefore, HPC systems are usually equipped with uninterruptible power supply (UPS) to handle accidentally power supply failure. Second, a single HPC system consumes a lot of power. The target exascale system expects to draw power at 20-30 MW. Such a high amount of power consumption requires expensive infrastructure and advanced uninterrupted power supply services. Third, a lot of top powerful supercomputers are charged by peak power consumption instead of actual energy consumption in U.S [87]. For example, from Clausen et al [22]'s report power supply institutes charge the supercomputers in national laboratories (*National Center for Supercomputing Applications (NCSA) and Lawrence Livermore National Laboratory (LLNL)*) based on its peak power consumption instead of actual energy consumption. In such cases, energy efficient computing fails to save operating cost. Forth, the power consumption in HPC varies between seasons, days and nights, and different jobs. The power variation increases the difficulty and additional costs to power supply institutions and power grid operators [31, 54]. Power grid operators even consider charging penalties from data center owner [21].

As the limitation of power supply infrastructure and giant operating cost, power was identified as one of the most critical challenges to achieve exascale computing [64]. To increase the power efficiency of future supercomputers, the U.S DoE sets the goal to achieving exascaling computing at 20 MW at the beginning and loose the requirement to 20-30 MW later. With today's technology, the estimated power consumption of exascale supercomputers will be 50-60 MW. In order to meet the DoE goal, research of power efficient computing conducted in all areas – hardware, system, and

workloads.

Power bounded computing schedules power to hardware resources with the objective of maximizing application performance or system throughput, at the same time enforces the power bounds on the system. Power bounded computing distinguishes from power aware techniques from multiple ways. First, power bounded computing focuses on the system level to maximize system performance under a power bound. Even though, power bounded computing is not isolated from hardware techniques. Contrarily, power bounded computing exploits new hardware's advantages and improve workloads performance according to workload characteristics. Second, power bounded computing has different goals compared with power aware computing. Power bounded computing brings the energy efficient and power aware techniques including hardware overprovision, power shifting and workload execution control under the goal: maximally transform power budget as application performance.

As the importance of a power bounded system for HPC, more and more research innovations come out recently. Rountree el al. [88] first propose RAPL as an alternative to DVFS to enforce power bounds in the HPC environment. DVFS is able to save power consumption or increase energy efficiency but is hard to keep the power always below a power bound as stable as RAPL. Patki et al. [84] demonstrate hardware overprovision policy has the potential to improve system throughput under power bounds. As applications consume different power on a node especially under different configurations, keeping the power budget on a node only under TDP is quite unreasonable approach and would waste a lot of power budget. Instead, supplying just enough power to a node and activating more hardware with enforcing power bound on the system improves system throughput. Patki et al. [82] develop RMAP, a resource manager for power-constrained systems based on their previous work [84]. Observed by Inadomi et al. [52], the hardware variability will be exacerbated in a power bound. Inadomi et al. [52] analyze the variation between nodes and propose a variation-aware power budgeting scheme to increase application performance.

Our proposed works [42, 44, 106, 108] discussed in Chapter 3, 4, 5 differs from these studies in three main aspects. First, our study reveals the different impacts on applications with different types of scalability. Such findings are helpful for designing power management methods. Second, our approach considers both node-level and cluster-level techniques and integrates them tightly to improve application performance on NUMA multicore-based systems under power bound. Third, our approach considers co-scheduling hardware and power resource with workloads to best match

resource and workloads under a power bound.

2.2.4 Power Bounded Heterogeneous Computing

The race for high performance and power efficient computing drives the integrating of accelerators (GPUs and FPGAs) in data centers and supercomputers. As a result, an increasing ratio of supercomputers integrated the GPU accelerators to increase system computational capability. From the trend of top 500 supercomputers, it is predictable that heterogeneous systems with GPU accelerators will be more common in HPC systems. The deployment of accelerators turns high performance computing as heterogeneous computing.

To improve power efficiency for heterogeneous supercomputers, researchers have utilized DVFS, co-scheduling to exploit the accelerators' advantage. On GPU architecture, researchers mainly rely on DVFS to reduce power and save energy consumption [70]. Ge et al. [43] experimentally study the impacts of DVFS to GPU application performance and GPU power consumption on a NVIDIA K20 GPU. Ge et al. [43] also compares the difference of impacts of DVFS on GPU and CPU. You and Chung [102] propose a DVFS framework to coordinate embedded GPUs power consumption with minimal performance degradation in mobile systems. Harmonia coordinates the hardware states of GPU core and GPU memory based on sensitivity predictors to save energy consumption by GPGPU applications [86].

DVFS is also used to increase energy efficiency or power cap on a CPU-GPU system. Chau et al. [16] utilize DVFS for CPU-GPU system and achieves near-optimal energy efficient job scheduling. Mei el al. [69] save 30∼36% energy for real-time tasks on CPU-GPU hybrid clusters with heuristic scheduling algorithm. Zhu et al. [105] first identify a workload is better to run on CPU or GPU, and co-schedule jobs on the two architectures with DVFS coordinating on both GPU and CPU to meet a power cap.

Power and performance modeling for GPU is essential to implement power bounded computing on GPU architecture. Hong and Kim [50] identify the memory bandwidth may limit application performance and propose an integrated power and performance (IPP) prediction model to find out an optimal number of activated processors for GPU applications. Guerreiro et al. [47] estimate GPU power consumption accurately across multiple DVFS configurations for different generations GPUs with the information of application GPU usage patterns. Majumdar et al. [66] develop performance and power prediction models and adjust GPU frequency for upcoming kernels by using

Model Predictive Control.

The work presented in Chapter 6 differs from previous research in multiple ways. First, the goal of our research is to maximize system throughput or application performance under system power bounds. Second, we utilize the resource utilization trace as application performance indicator to monitor application running statues in real time. Third, we exploit machine learning methods to dynamically predict the application performance while adjust hardware configuration and validate with resource utilization trace. Forth, we estimate and deploy the optimal configurations online without offline profiling or code intrusion.

2.3 Problem Statement of Power Bounded Computing

Amdahl's Law is a parallel speedup model commonly used by the researchers. For parallel computing, the increased number of nodes, the increased number of cores, and higher core frequency are both considered as the enhancement. Originally, the speedup is defined as the ratio of sequential to parallel execution time [40]:

$$S_N(w) = \frac{T_1(w)}{T_N(w)} \tag{2.1}$$

where,

w: the workload,

$T_1(w)$: the sensational execution time to complete workload W with 1 node.

$T_N(w)$: the parallel execution time to complete workload W with N nodes.

In multicore and many core era, the speedup model can be re-defined as:

$$S_{N\times K}(w) = \frac{T_1(w)}{T_{N\times K}(w)} \tag{2.2}$$

where,

$T_{N\times K}$: the parallel execution time to complete workload W with N nodes, and each node activating K cores.

In power bounded clusters, the system may have to adjust the core frequency to meet the power budget requirement. The model to describe the relationship between speedup and frequency can be denoted as:

$$S_{N\times K}(w, f) = \frac{T_1(w, f_{base})}{T_{N\times K}(w, f)} \tag{2.3}$$

where,

f: the clock frequency

$T_1(w, f_{base})$: the sequential execution time to complete workload w on 1 core with base frequency.

$T_{N\times K}(w, f)$: the parallel execution time to complete workloads with $N \times K$ cores for frequency f. Therefore, for power bounded computing, we need to consider all three factors simultaneously to maximize application performance under a power bound.

In a power constrained environment, both power budget and hardware are limited resources. We assume the system power budget as P_b, and the system hardware machine(s) as M. The object of power bounded computing is to maximize application performance and system throughput under the system power budget P_b with available hardware machine(s) M. Therefore, we can formulate the power bounded computing problem as follows:

Given a job queue Q with a list of parallel workloads $\{W_1, W_2, ..., W_n\}$, a machine M and a total power bound P_b, find an optimal power, hardware and job allocation such that:

1. $Throughput(a^*, Q, M) = max_{\alpha\in A} Throughput(\alpha, Q, M)$, and
2. $\sum_{m\in M} P^*_m \leq P_b$

Here, α is a power, hardware and resource allocation tuple (J, Hw, P) and A is the space that comprises all possible values of α. The hardware allocation can distribute multiple levels, from power boundable component to a full node. We define a component as power-boundable if the component can and will always operate under the power cap allocated to it.

As described in Chapter 1, we tackle the power bounded computing from node level to cluster level, from node monopolize systems to node sharing systems, from homogeneous systems to heterogeneous systems. Similarly, we can view the problem from different perspectives.

2.3.1 Problem Statement on Node Level

On the node level, we assume the jobs does not share the single node. Therefore, the throughput maximization problem is equal to maximize application performance of each workload $W|W \in \{W_1, W_2, ..., W_n\}$ in the job queue Q. We formulate the problem of power bounded computing on node level as follows:

Given a parallel workload W, a node M comprising a set of K power-boundable components C_1, C_2,...,C_K, and a total power bound P_b, find an optimal power allocation tuple $a^* = (P_1^*, P_2^*, ..., P_K^*)$ such that:

1. $perf(a^*, W, M) = max_{\alpha \in A} perf(\alpha, W, M)$, and
2. $\sum_{i=1}^{K} P_i^* \leq P_b$

Here, α is a power allocation tuple and A is the space that comprises all possible values of α. Therefore, the objective of power bounded computing on node level is to identify a power allocation strategy between components. The power allocation should meet the applications demands on each component and maximize application performance under the power budget.

2.3.2 Problem Statement on Cluster Level

In a node monopolize cluster, the system can adjust the number of participated nodes to trade-off application performance and power consumption. Meanwhile, the system can also change the cores utilized on each node to control performance achievement and power consumption. Without a power budget, on the cluster level, the application scalability is determined by the contention of shared resource such as network bandwidth. On the node level, the memory hierarchy contention impact application performance significantly. Under a given power bound, the system needs to rethink the resource activation and power allocation on both node level and cluster level. Here, we formulate the system power bounded scheduling in a cluster as follows:

Given a parallel workload W, a homogeneous cluster M with N nodes. Each node comprises a set of K power-boundable components C_1, C_2,...,C_K. The cluster shall operate under a total power bound P_{ub}, find an optimal power allocation tuple $a^* = ((1 \times P_1^*), ..., (1 \times P_K^*), (2 \times P_1^*), ..., (2 \times P_K^*), ..., (N \times P_1^*), (N \times P_K^*))$ such that:

1. $perf(a^*, W, M) = max_{\alpha \in A} perf(\alpha, W, M)$, and
2. $\sum_{i=1}^{N} \sum_{j=1}^{K} P_{ij}^* \leq P_b$

Here, α is a power allocation tuple and A is the space that comprises all possible values of α. In such a system, there are overall $N \times K$ configurations for system to explore. It is not possible and not necessary to profile every configuration to identify the optimal configurations that maximize

performance under the system power bound. To simplify the configuration search, we assume the application performance can be estimated as:

$$perf_{(n,k,freq)} = f(n, k, freq, perf_{base}) \tag{2.4}$$

$perf_{base}$ is the application performance runs with a single core at the base core frequency in a single node. Obviously, to accurate identify the best performance, the model must identify the relation between performance and core frequency, core number and node number precisely. In this dissertation, we run application independently in a cluster, thus the benchmark can exclusively utilize the system network bandwidth. Therefore, We can simplify the problem by utilizing a linear function to describe the relationship between application performance and number of nodes. As a result, we focus on the relationship between application performance and core frequency and activated cores number. The core frequency and activated cores number must ensure the power consumption is lower than the given power bound. We can assume the power model as:

$$power_{(n,k,freq)} = f(n, k, freq, power_{base}) \tag{2.5}$$

$power_{base}$ is the application power consumption while executing application with a single core at the base core frequency in a single node. From related work and our experimental observation, the power is linearly related with number of nodes, number of cores and core frequency. While we control the variable: number of nodes and number of cores, we can estimate the core frequency according to available power budget and further predict the application performance.

2.3.3 Problem Statement on Node Sharing Systems

For node sharing systems, the scheduler needs to consider performance interference between jobs that co-running on a node. The performance interference is caused by memory contention and power budgeting. Even when a job is finished and a new job starts to execute, the system shall keep the power consumption under the given power bound. We formulate the power bounded computing for node sharing systems as follows:

Given a job queue Q with a list workloads $W|W \in \{W_1, W_2, ..., W_n\}$*, a homogeneous cluster M with N nodes. The cluster shall operate under a total power bound* P_b*, find an optimal job, power*

and hardware resource co-scheduling tuple a^* such that:

1. $Throughput(a^*, W, M) = max_{\alpha \in A} Throughput(\alpha, W, M)$, and
2. $\sum_{i=1}^{N} \sum_{j=1}^{K} P_{ij}^* \leq P_b | \, for \; t \in (start, end)$

Chapter 3

Cross Component Power Coordination

High performance power bounded computing is built upon the premise that every compute node in the system can and will operate under a given power budget. To enable the power bound on a large scale system, each node has to set a power cap. By bounding per node power consumption, a large-scale system is capable of reconfiguring itself according to its current workload to achieve better performance under the same power budget.

Intelligently coordinating power among computer components is the key to power bounded computing. To achieve optimal performance, the participating components must work in concert and match the applications' demands. Meanwhile, unused components can transition into power saving states, allowing more power to allocate to activated components for better performance.

Power consumption on the node level can be divided as static power and dynamic power. The static power refers to the power consumption while only the operating system and basic services are running. Meanwhile, the dynamic power refers to the power consumption increment when users have submitted jobs that are running on the node. The system may not be able to cut static power and may impact application performance significantly if the components power budget is lower than the static power. Therefore, power allocation primarily focuses on dynamic power distribution.

To gain insight on the dynamics between power allocation and application performance without being distracted by the complexity of coordinating multiple closely interacting components,

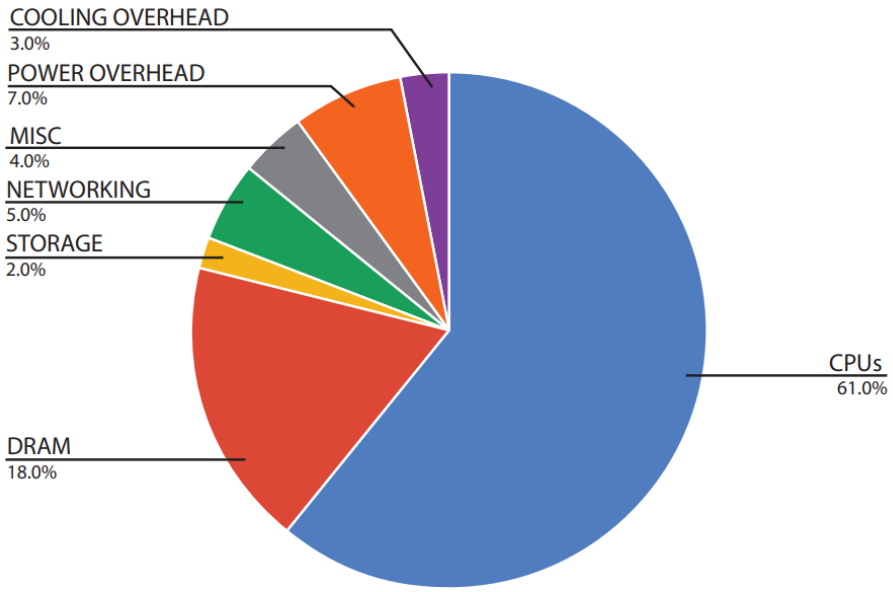


Figure 3.1: Approximate distribution of peak power usage by hardware subsystem in a modern data center. The figure assumes two-socket x86 servers and 12 DIMMs per server, and an average utilization of 80%. The figure is copied from Barroso et al. [8].

we focus on the problem of power bounding on the processor cores and DRAM modules. We make this simplification based on two observations. First, processor cores and memory modules dominate node power consumption on current and emergent HPC systems as seen from Figure 3.1. Second, shifting power between CPU and memory affects both the CPU operating speed and data access rate, the two primary parameters that jointly determine the performance of HPC applications.

Undoubtedly, high performance power bounded scheduling on the node level is a challenging problem because its implementation involves cross-layer, cross-component scheduling decisions that must be tailored to workload-specific characteristics like computation and data access patterns. To design a robust and effective scheduler, we must have a deep and accurate understanding of the intricate relationship between hardware, workload, performance, and power.

Incorporating NUMA architecture into power bounded computing brings in new opportunities to reduce power, increase performance, or achieve both. However, effective power coordination on NUMA multicore systems can be challenging. First, each application has its own unique workload characteristic (like compute to memory access ratio, local/remote memory accesses, and concurrency profile) which subsequently creates specific power demand from each component. Second, complex relations exist between application workload, system resource, and power allocation. An effective

power allocation schema must fully evaluate the interactions between the major parts of the system and allocate the correct amounts of power to the right components at the right time.

This chapter studies the problem of coordinated power allocation between processors and memory modules on power bounded systems. We experimentally and analytically investigate the dynamics between cross-component power allocation and application performance, identify the patterns of power allocation scenarios, and develop optimal power allocation methods. Section 3.1 investigates the impact of power bounds and summarizes the critical power levels for cross component power coordination. Section 3.1 proposes the heuristic power allocation methodology to exploit the application power levels and shows the experimental results. Section 3.2 further considers other application characteristics (scalability and memory access intensity) besides application power levels, and introduces the application aware power allocation method and discusses the results elaborately. Finally, Section 3.3 summarizes the contribution of proposed methodology.

3.1 Heuristic Power Allocation

3.1.1 The Impact of Power Bounds

To examine the technical feasibility and performance impact of cross-component power coordination, we run a series of experiments on an IvyBridge server. The architectural details of the server is listed in the Appendix. In these experiments, we capped the power on CPU and DRAM components using the RAPL to meet the specified power budget.

Figure 3.2 shows the experimental results of different power allocations between CPUs and DRAMs under a combined power budget of 140 Watts for the HPCC Star Random Access benchmark. In the experiment, we run HPCC star Random Access with 20 MPI process. Each process conquers one core of the 20 IvyBridge cores. The description and inputs of the benchmark is listed in Table 3.2. The results reveal that power allocation between components has a significant impact on application performance. A poorly coordinated power allocation could decrease the application performance up to 30× compared to the most optimal power coordination. Meanwhile, poorly coordinated power distribution between components does not use the power budget efficiently, wasting valuable resources needed by other nodes in the HPC system.

Figure 3.3a and Figure 3.3b shows the performance impact with different power allocations under a combined power budget from 72 watts to 180 watts for HPCC DGEMM and Star Random

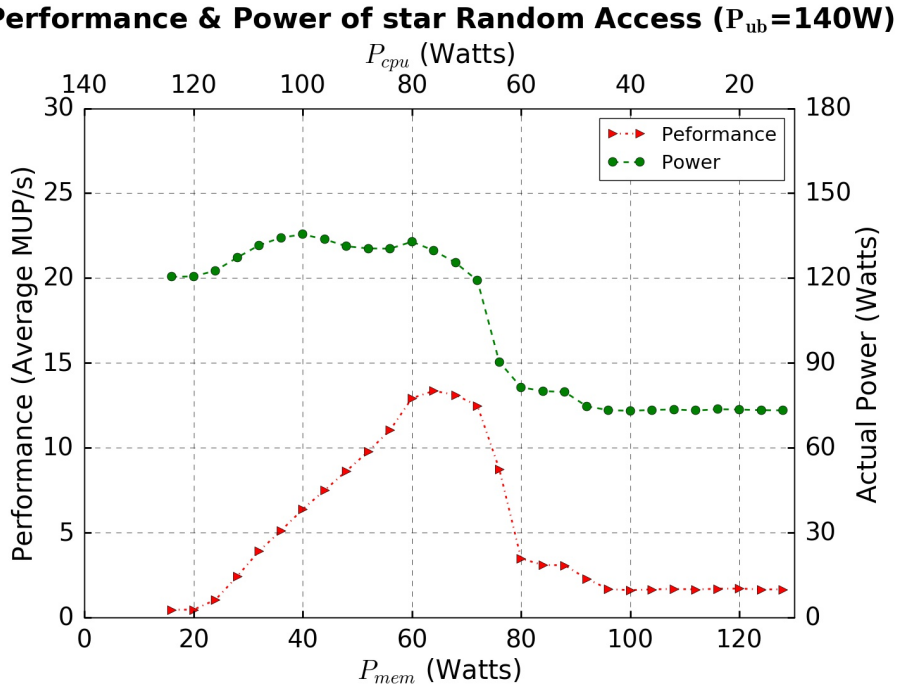
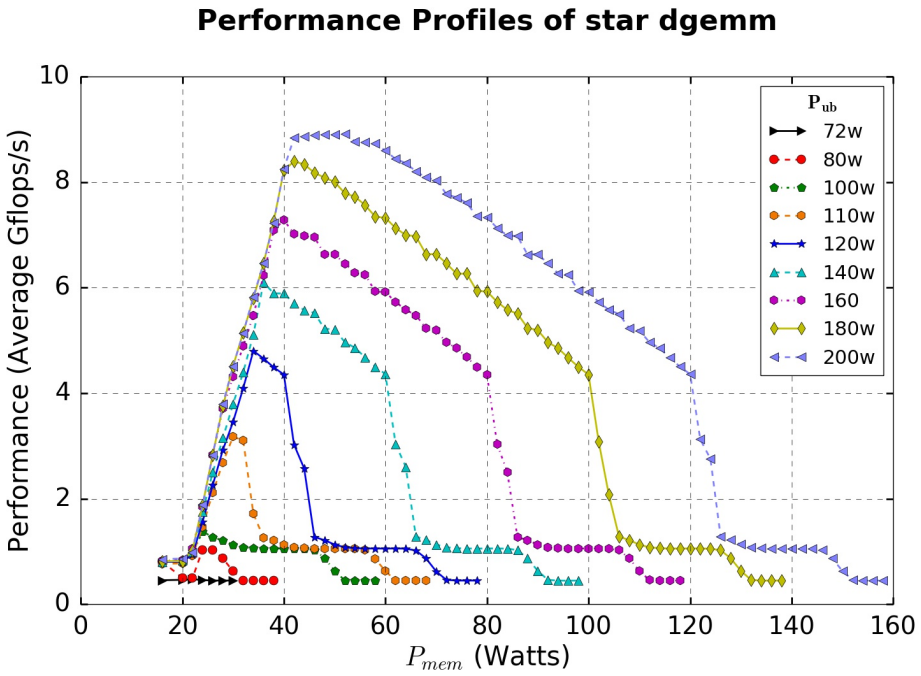
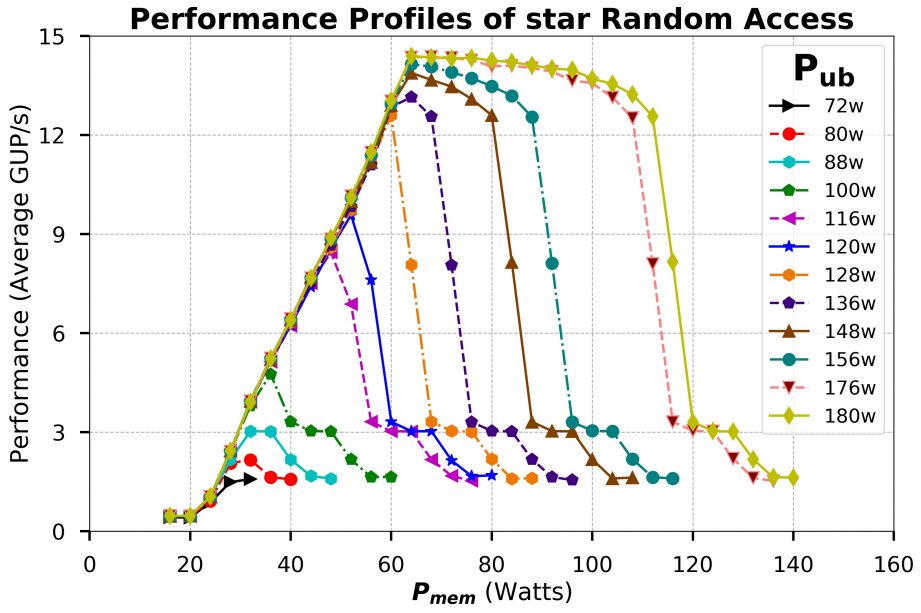


Figure 3.2: The impact of power allocation on application performance and power consumption of the HPCC Star Random Access benchmark. The performance units (UP/s) means number of updates per second. The top and bottom x axes designate the power budgets of processors and memory respectively.

Access respectively. As seen from the figure, we can observe that: first, the impacts of power allocation on application performance are non-linear but have clear patterns. Second, the optimal power allocation relies on the application characteristics. The power allocation strategies must understand the patterns for each application and decide the distribution according to system power budgets. We also note that the performance metric, $perf$, can have different definitions depending on both the application and the user's demand. As seen from Figure 3.3a and Figure 3.3b, the optimal power allocation is determined by both workload characteristics and available power budget. We can clearly split the power performance relationship into multiple scenarios and identify the optimal power allocation with the scenario analysis. Section 3.1.2 summarized the scenarios of power allocation under different power budgets.



(a) HPCC Star DGEMM.



(b) HPCC Star Random Access.

Figure 3.3: The performance impact of cross-component power allocations on HPCC Star DGEMM (a) and Star Random Access (b) benchmarks. The experiments are evaluated on a node with two 10-core IvyBridge processors. Each MPI process occurs in one core. Table 3.2 listed the detail parameters.

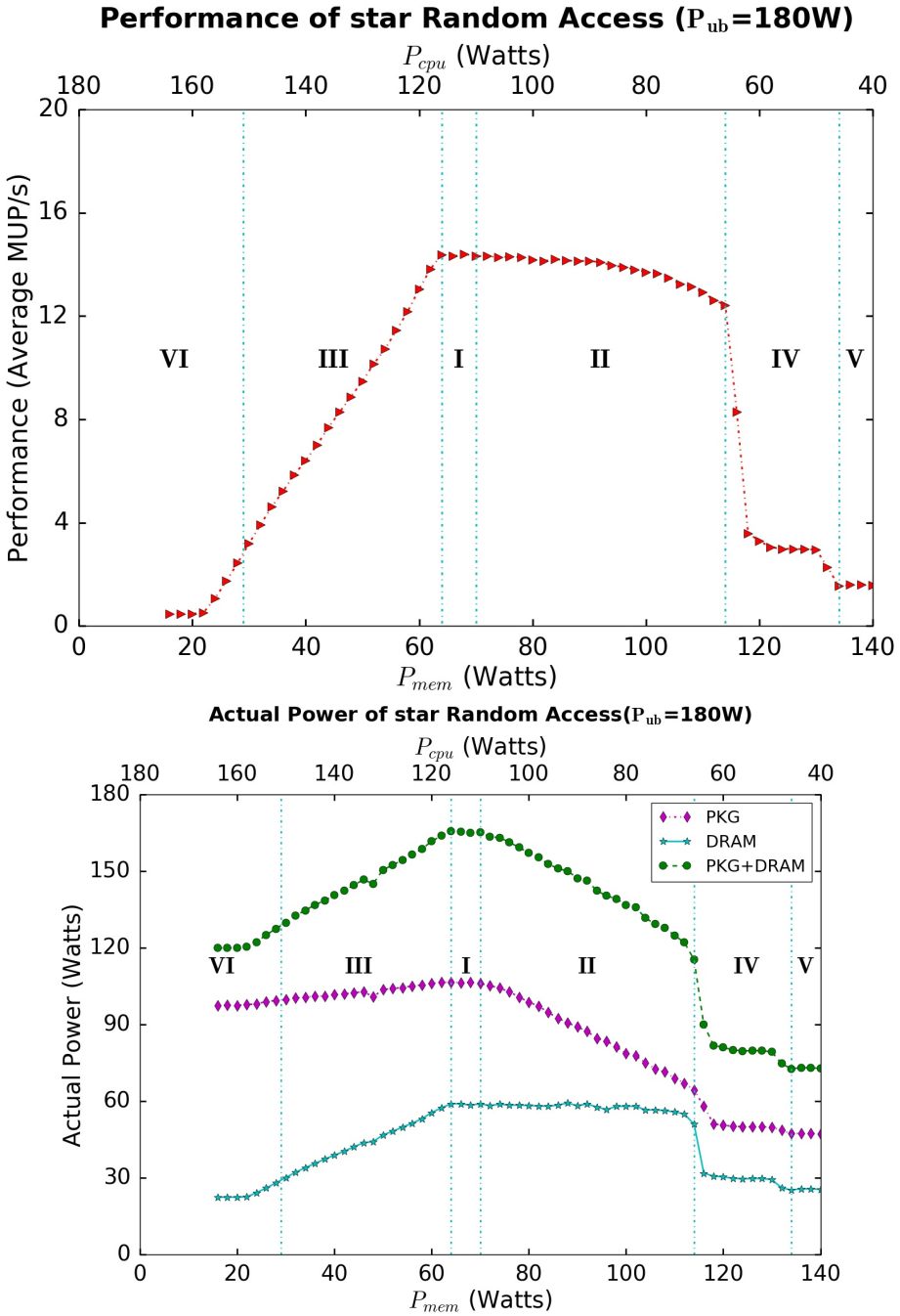


Figure 3.4: Categorization of power allocation scenarios. The plots of application performance (a) and actual power consumption (b) for different power allocations between processors and memory modules visually reveal six categories of power allocation scenarios.

3.1.2 The Scenarios of Power Allocation

By plotting the curves of application performance and the actual power of major components, we identified that the impacts of power allocations fall into six categories. We utilize the results by running HPCC Star Random Access under the power bound of 180 watts as an example. The six categories of power allocation scenarios described as follows:

(I) *Adequate power allocation for both CPUs and memory.* The power allocation $P_{\texttt{cpu}} \in [110, 116]$ (Watts) or $P_{\texttt{mem}} \in [64, 70]$(Watts) in Figure 3.4 falls into this scenario. In scenario I, the power caps on both CPU and memory exceed their maximum power demands. Therefore, the power caps of both components do not impact the application performance.

(II) *Adequate memory power, lightly constrained CPU power.* The power allocation $P_{\texttt{cpu}} \in [66, 110]$ (Watts) in Figure 3.4 falls into this scenario. In scenario II, the power caps are not able to operate all processor cores running at the highest performance state, but stay still at a performance state. As CPU power budget decreases, application performance decreases monotonously. On the other side, the power budget for DRAM is higher than the maximum power of DRAM. Scenario II is not an ideal allocation in this example because memory wastes its power budget while CPU performance is constrainted by available power.

(III) *Adequate CPU power, constrained memory power.* The power allocation $P_{\texttt{mem}} \in [29, 64]$ (Watts) in Figure 3.4 falls into this scenario. In scenario III, the memory cap limited the memory performance, and further the application performance; thus increasing the power budget of DRAM dramatically improves application performance. Combined with these scenarios, we identified that application performance is much more sensitive to memory power constraint than CPU power constraint.

(IV) *Adequate memory power, seriously constrained CPU power.* The power allocation $P_{\texttt{cpu}} \in [48, 66]$ (Watts) in Figure 3.4 falls into this scenario. In scenario IV, CPU power is significantly under-budgeted, and the application performance decreases sharply from scenario II.

(V) *Adequate memory power, minimum CPU power.* The power allocation $P_{\texttt{cpu}} \in [-, 44]$ (Watts) in Figure 2 falls into this scenario. In scenario V, the actual CPU power levels off and stays at 44 Watts, even when a lower power budget is allocate to CPUs. The corresponding actual DRAM power consumption is also constant at 26 Watts.

Table 3.1: Optimal allocation and critical component vs. power budget.

P_{ub}	Valid Allocation Scenarios	Optimal Allocation		
		Intersection of		Critical Comp.
large	I, II,III,IV,V,VI	I		none
↓	II,III,IV,V,VI	II	III	DRAM
↓	III,IV,V,VI	III	IV	CPU
↓	IV,V,VI	IV	V	DRAM
small	V,VI	V	VI[1]	CPU

(VI) *Adequate CPU power, minimum memory power.* The power allocation $P_{\texttt{mem}} \in [10, 29]$ (Watts) in Figure 2 falls into this scenario. In scenario VI, DRAM consumes a constant power of 22 Watts even when a lower budget is allocated to DRAM. This scenario cannot ensure the system power bound and often delivers the worst performance.

Table 3.1 summarizes the location of the optimal allocation for varying power budgets. From the optimal cross-component power allocation, shifting a small amount of power to either CPUs or DRAM could decrease performance. However, shifting power to CPUs could degrade performance more significantly than DRAM.

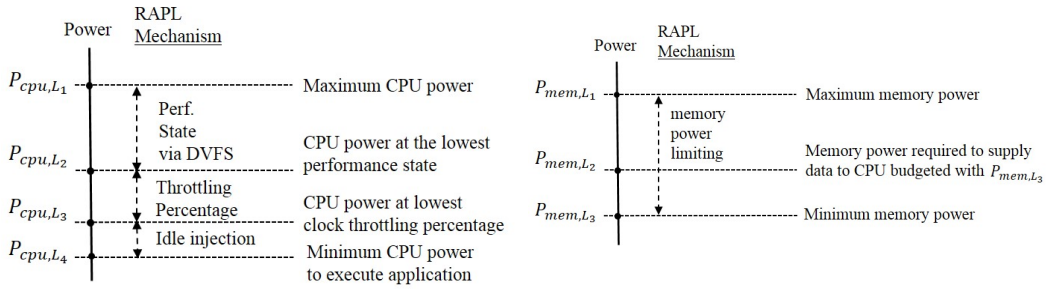
3.1.3 Linkage Between Scenarios and Critical Power Levels

In Figures 3.2, Figure 3.3a and Figure 3.3b, we observe that each power allocation category is delimited by a pair of critical component power values. For the experimental platform in this work, there are four critical processor power values ($P_{\texttt{cpu},L_i \; for \; i=1..4}$) and three critical memory power values ($P_{\texttt{mem},L_i \; for \; i=1..3}$)) as shown in Figure 3.5. These application-specific values define boundaries of power allocation scenarios and correspond to the transition points at which RAPL switches from one power capping mechanism to another. Specifically, the seven critical power values are described as follows:

$P_{\texttt{cpu},L_1}$**:** the maximum processor power consumption. Processor runs at its nominal frequency and highest performance state.

$P_{\texttt{cpu},L_2}$**:** the processor power consumption when operating at the lowest performance state. $[P_{\texttt{cpu},L_2}, P_{\texttt{cpu},L_1}]$ forms the power range of processor performance states.

$P_{\texttt{cpu},L_3}$**:** the power consumption when the processor is at the lowest percentage of clock throttling on the system.



(a) Critical Power Levels of Processor

(b) Critical Power Levels of Memory

Figure 3.5: Critical power values for CPU and memory and their relations with RAPL power limiting mechanism.

$P_{\texttt{cpu},L_4}$: the minimum power consumption when the processor is actively executing applications. If the processors are imposed with a budget that is lower than $P_{\texttt{cpu},L_4}$, they still consume $P_{\texttt{cpu},L_4}$ watts of power. $P_{\texttt{cpu},L_4}$ is the same for all applications and controlled by hardware.

$P_{\texttt{mem},L_1}$: the highest DRAM power consumption when both CPUs and DRAM operate at the highest performance state to execute the application.

$P_{\texttt{mem},L_2}$: the corresponding DRAM power when there is no power cap for DRAM and processor power is at $P_{\texttt{cpu},L_3}$.

$P_{\texttt{mem},L_3}$: the minimum DRAM power consumption set by the hardware for a running system. If DRAMs are imposed with a budget that is lower than $P_{\texttt{mem},L_3}$, they still consume $P_{\texttt{mem},L_3}$ Watts of power. This minimum power is the same for all applications.

3.1.4 Category-Based Heuristic Power Coordination

The existence of critical power levels provides two important heuristics. First, the power budget allocated to a computer system should be greater than $P_{\texttt{cpu},L_2} + P_{\texttt{mem},L_2}$ to operate in a productive manner. Second, if the power budget is above the threshold, the critical power values dictate the set of valid power allocation scenarios and corresponding optimal cross-component allocation.

Algorithm 1 shows the category-based power coordination method based on previous analysis. Essentially, this method breaks the set of possible power budgets into four subsets:

- (A) adequate power for both components to operate at the highest performance state,
- (B) adequate power only for one component to operate at the highest performance state,

- (C) neither component has adequate power to run at its highest performance state,
- (D) both components must be throttled down to satisfy the power limit.

Then for each subset, this method first allocates a minimally adequate power to the critical component and then gives the remaining budget to the other component.

Algorithm 1 CORD: Category-Based Heuristic Power Coordination

procedure CORD(P_{ub})
  $status \leftarrow Success$
  if $P_{ub} \geq P_{\texttt{cpu},L_1} + P_{\texttt{mem},L_1}$ **then**
    $P_{\texttt{cpu}} \leftarrow P_{\texttt{cpu},L_1}$
    $P_{\texttt{mem}} \leftarrow P_{\texttt{mem},L_1}$
    $status \leftarrow Hint:$ *power surplus* $(P_{ub} - P_{\texttt{cpu},L_1} - P_{\texttt{mem},L_1})$!
  else if $P_{ub} \geq P_{\texttt{cpu},L_2} + P_{\texttt{mem},L_1}$ **then**
    $P_{\texttt{mem}} \leftarrow P_{\texttt{mem},L_1}$
    $P_{\texttt{cpu}} \leftarrow (P_{ub} - P_{\texttt{mem}})$
  else if $P_{ub} \geq P_{\texttt{cpu},L_2} + P_{\texttt{mem},L_2}$ **then**
    $P_{\texttt{cpu}} \leftarrow P_{\texttt{cpu},L_2}$
    $P_{\texttt{mem}} \leftarrow (P_{ub} - P_{\texttt{cpu}})$
  else
    $P_{\texttt{mem}} \leftarrow min(P_{ub}, P_{\texttt{mem},L_2})$
    $P_{\texttt{cpu}} \leftarrow P_{ub} - P_{\texttt{mem},L_2}$
    $status \leftarrow Warning:$ *budget too small*!
  end if
  return $(P_{\texttt{cpu}}, P_{\texttt{mem}}, status)$
end procedure

3.1.5 Experimental Results and Discussion

We experimentally evaluate the problems of heuristic cross-component power coordination on the node with IvyBridge processors. The benchmarks used for evaluation include: HPC Challenge Benchmark (HPCC) [65], NAS Parallel Benchmarks (NPB) [7], and UVA STREAM in our experimental study. These benchmarks, listed in Table 3.2, consist of eight kernels (SRA, STREAM, DGEMM, IS, EP, CG, MG, and FT) and three pseudo-applications (BT, SP, and LU) and test a range of computation and memory access patterns in typical scientific workloads. In each experiment, we use all physical CPU cores to run one benchmark with one core being mapped with one MPI process or one OpenMP thread.

Figure 3.6 shows the performance and power profiles of two NPB applications (SP and BT) and verifies the patterns and categorizations described in Section 3.1.2. The experimental results

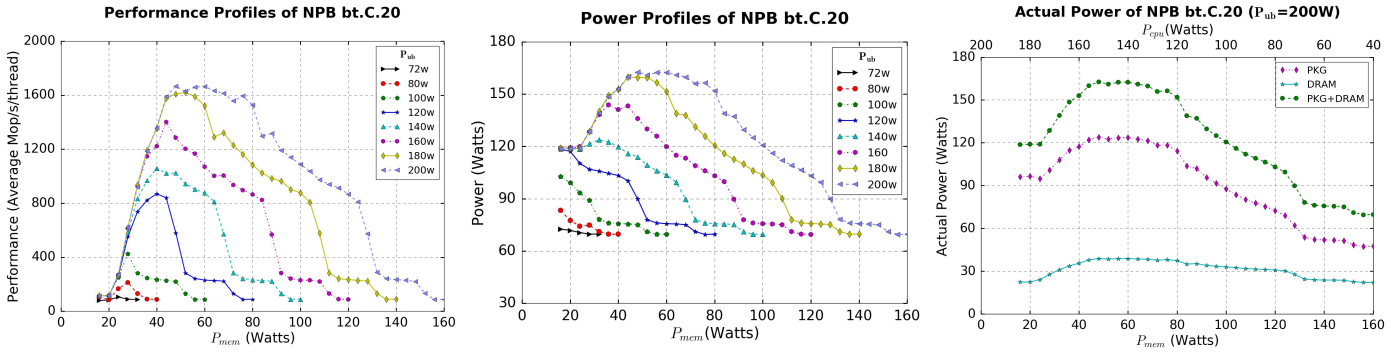
Table 3.2: List of benchmarks used in heuristic power coordination

Benchmark	Problem size/Input	Description and Workload Pattern
SRA	2^{25} words	Embarrassingly parallel, random memory access
STREAM	Array Size = 8.9e6	Synthetic benchmark, measuring memory bandwidth
DGEMM	Ns =32000	Double precision matrix multiplication, compute intensive
BT	C	Block Tri-diagonal solver, compute intensive
SP	B	Scalar Penta-diagonal solver, compute/memory
LU	C	Lower-Upper Gauss-Seidel solver, compute/memory
EP	C	Embarrassingly Parallel, compute intensive
IS	C	Integer Sort, random memory access
CG	C	Conjugate Gradient, compute/irregular memory access
FT	C	Discrete 3D fast Fourier Transform, compute/memory
MG	B	Multi-Grid on a sequence of meshes, compute/memory

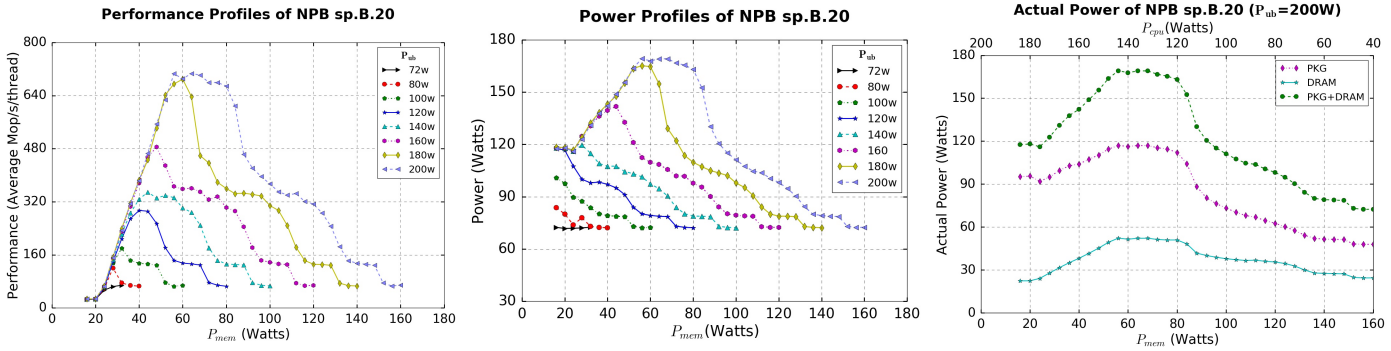
show that there are both universal patterns and workload-specific features with different power allocations under a total budget. We summarized two key findings here. First, computation and memory access patterns determine the shape of the performance-power allocation curves. Generally, memory intensive workloads demand more power budget for memory components. Conversely, computing intensive workloads require less power budget for memory but more power for processors. Second, the variations of workload characteristics over time impact the regularity of the performance-power allocation curves in each scenario category. For example, pseudo-applications like BT and SP may comprise multiple memory access patterns during their execution.

To evaluate the accuracy of the proposed heuristic power allocation method, we compared the selected distribution $\alpha = (P_{\texttt{cpu}}, P_{\texttt{mem}})$ by the coordination algorithm against the exhaustive searched optimal distribution $\alpha^* = (P^*_{\texttt{cpu}}, P^*_{\texttt{mem}})$. The experimental results are shown in Figure 3.7. We summarized the performance of Algorithm **CORD** as follows:

First, when the power budget is higher than $P_{\texttt{cpu},L_1} + P_{\texttt{mem},L_2}$, there are noticeable differences between α and α^* as shown in Figure 3.7a. In general, Algorithm **CORD** consumes less power compared with the optimal configuration by exhaustive search. The reason for this is that Algorithm **CORD** can determine the budget surplus while exhaustive search has no such capability. Second, when the power budget is lower than $P_{\texttt{cpu},L_1} + P_{\texttt{mem},L_2}$, α is the same or very close to α^* for most of the cases in the study. However, the shifts between α and α^* is 12 Watts at the $P_{\text{ub}} = 160$ Watts. Algorithm **CORD** allocates the maximum power budget to memory and runs lower CPU performance states. Overall, the proposed power coordination algorithm achieves or approaches

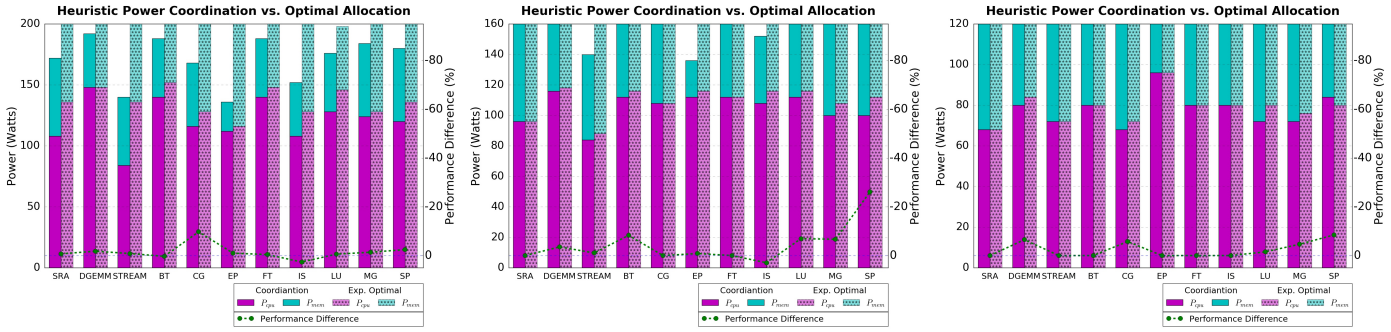


(a) NPB BT Performance (b) NPB BT Actual Power (c) NPB BT Component Power



(d) NPB SP Performance (e) NPB SP Actual Power (f) NPB SP Component Power

Figure 3.6: Performance and power profiles of compute-intensive and memory-intensive applications. While all benchmarks share similar patterns of allocation scenarios, each has application-specific patterns and power ranges for each component for the scenarios.



(a) $P_{\text{ub}} = 200$Watts (b) $P_{\text{ub}} = 160$Watts (c) $P_{\text{ub}} = 120$Watts

Figure 3.7: Comparison between heuristic power coordination and the optimal power allocation identified from experiments.

maximum performance. On average, the performance difference between α and α^* is 3.5% when the power budget is above the suggested threshold $P_{\text{cpu},L_2} + P_{\text{mem},L_2}$. These results indicate that the proposed power coordination method is accurate under a practical power budget.

3.2 Application Characteristics Aware Power Allocation

In Section 3.1, we proposed heuristic power allocation to coordinate power between processors and memory under a power bound. This section will investigate the power bound impact on non-uniform memory access (NUMA), a widely deployed memory architecture in modern multicore systems. Incorporating NUMA architecture into power bounded computing brings in opportunities and challenges to reduce power and/or increase performance. The workload characteristics (like computer to memory access ratio, local/remote memory accesses, concurrency profile, etc) create specific power demand from each component. Therefore, the complexity relations between application workload and system resource increase significantly. An effective power allocation schema must fully evaluate the interactions between the application and major components of the system, as well as allocate reasonable resources at the right time. In this section, we will first discuss the workload characterization in Section 3.2.1, and propose the algorithm and corresponding framework in Section 3.2.2. In the end, we will evaluate the approach in Section 3.2.3.

3.2.1 Workload Characterization

To understand the applications' capability of exploiting NUMA architecture, we identified applications that show significantly different parallel scalability and memory access intensity. We labeled the parallel applications with one of the three values: low, moderate and strong based on their memory intensity or scalability. For example, as illustrated in Figure 3.8, DGEMM belongs to low memory intensity and high parallel scalability; STREAM belongs to moderate scalability and high memory intensity.

In order to more practically capture the impacts of different workload performance and power characteristics, we studied three metrics: parallel scalability, memory access intensity, and components' critical power levels.

3.2.1.1 Scalability

We define the performance scalability as the application performance changes with the number of cores (n) on a node activated to execute the workload. We applied the similar concept, power scalability, to describe how application power consumption changes with n. If we could build a model to estimate the performance and power scalability, we could use the model to find the

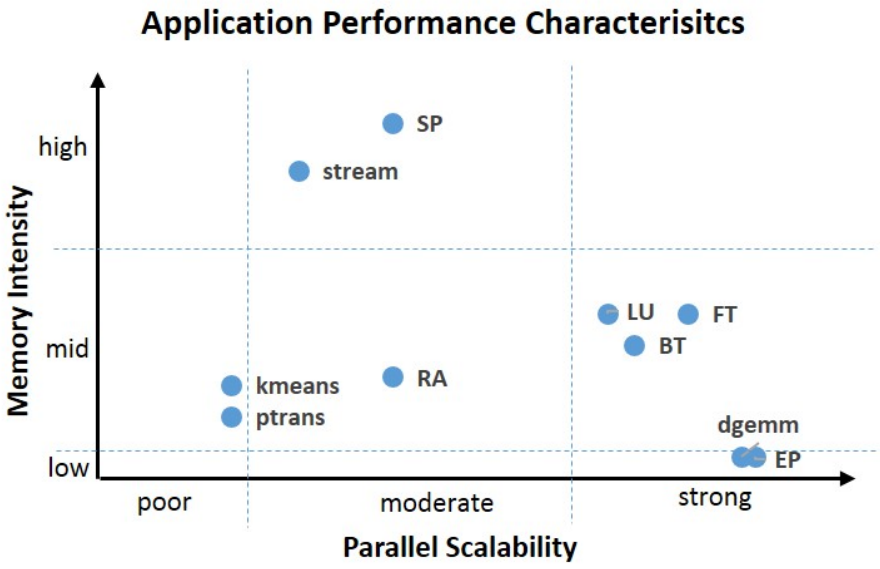


Figure 3.8: Application characteristics clustering. The scalability is measured on the two 12-core Haswell sockets node.

reasonable number of cores n that achieve optimal performance under a given power budget.

Empirically, performance $speedup(n) \propto n$ is true only for ideal or embarrassingly parallel applications; for most other parallel applications, $speedup(n)$ is a nonlinear function of n. For power consumption, $power(n) \propto n$ or is true for most parallel applications. This implies that we would need to profile a large number of execution configurations to accurately capture the scalability of parallel applications.

Fortunately, for most parallel applications, both performance and power scalability can be modeled by a segmented linear model, which comprises several segments and each segment has an approximately a linear relation to number of used cores. Figure 3.9 shows an instance by profiling applications SP and FT. We ran multithreads applications SP and FT with problem size C and each thread use 1 core. The experiment was executed on the two 12-core Haswell processors node as illustrated in the Appendix. FT is linearly scalable with the number of cores n for both performance and power. But for SP, performance scalability has to be described with a two segments model and power scalability can be approximated with one segments.

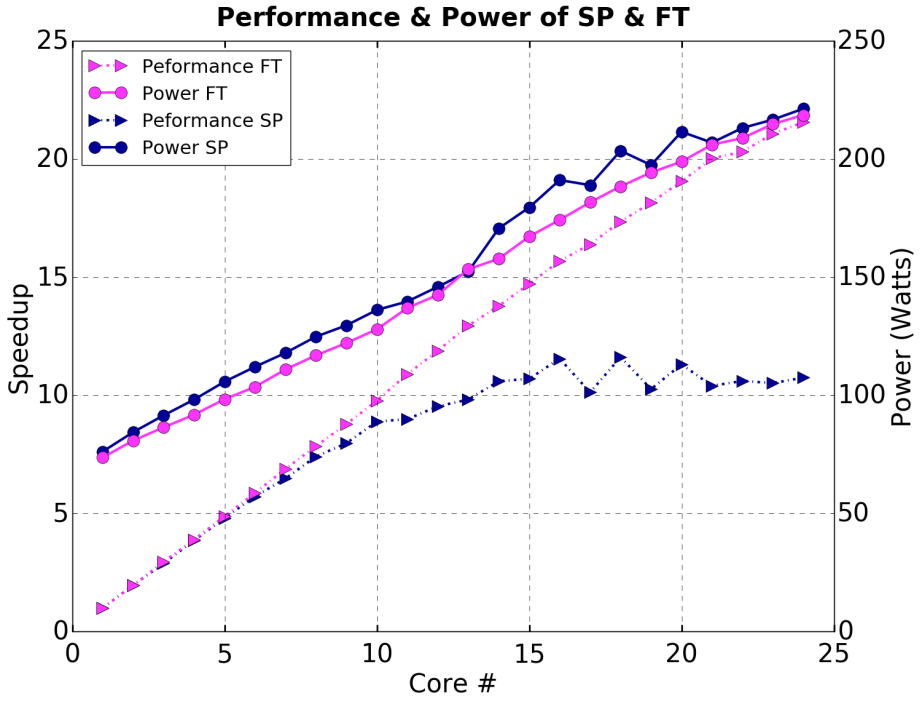


Figure 3.9: The performance and power of SP and FT. This figure shows (1) parallel scalability varies with applications; (2) segmented linear models can be used to approximate the scalability curves and estimate the number of activated cores for a given power bound.

3.2.1.2 Memory Access Intensity

Memory access intensity affects power bounded scheduling from three major aspects. First, the performance of memory intensive applications is more sensitive to memory power allocation. The performance of memory intensive applications may drop dramatically even with a slight reduction on memory power budget. This makes a sharp comparison with compute intensive applications, whose performance gradually decreases while CPU power allocation reduces. Such a difference is also partly due to the fact that CPU power has a larger range than memory power. To avoid adverse performance impact, power bounded allocation should ensure memory receive sufficient power, especially for memory intensive applications.

Second, memory intensive applications prefer an intelligent placement of activated cores and core affinity. Core placement and affinity affect memory contention and memory access costs due to the nature of NUMA architecture. On one hand, placing activated cores on the same socket may saturate memory buses and cause memory contention. On the other hand, distributing activated

cores among the sockets may increase remote memory access with longer lateness. An optimal core affinity makes a tradeoff between memory contention and remote accesses.

Third, core placement affects system power consumption; placing the activated cores on the same socket consumes less power if other inactivated sockets can transit to sleep states.

Figure 3.10 shows the impact of core affinity on performance and power for STREAM and EP on the Haswell Dual-processor node. STREAM is considered to be a memory intensive application while EP is compute intensive. Allocating 12 cores to STREAM threads and binding 6 to each processor socket delivers the maximum speedup and power efficiency, further increasing cores number only incurs higher power consumption without performance improvement. In comparison, binding all 12 threads to one processor delivers less STREAM performance. Compared with STREAM, EP doesn't show performance and power variation across core affinity.

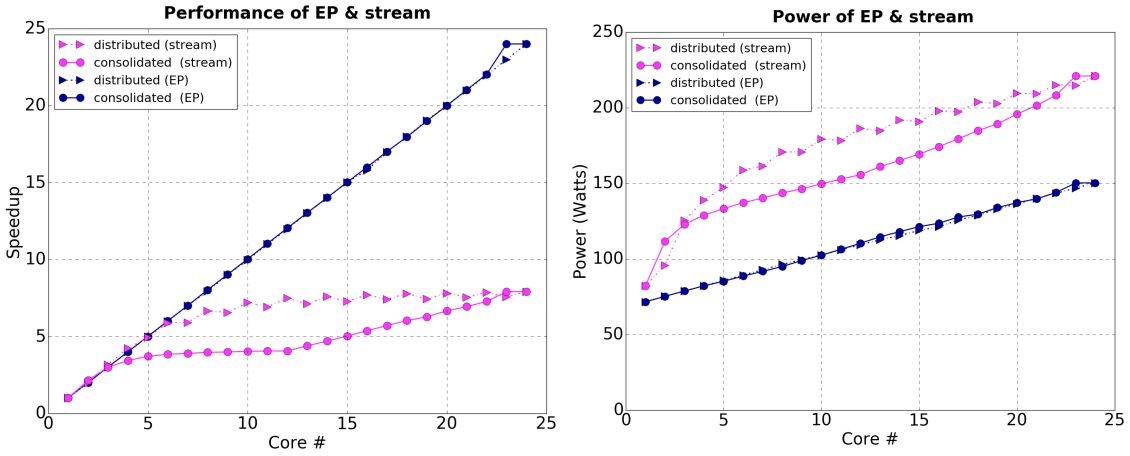


Figure 3.10: The impact of core affinity on performance and power. This figure shows: (1) the degree of impact differs between the compute intensive application and the memory intensive application; (2) for memory intensive applications, distributing cores to two sockets may result in better performance but a higher power than placing cores on the same socket.

3.2.1.3 Critical Component Power Levels

in Section 3.1, we have discussed how power coordination impacts application performance and actual power consumption. However, the power coordination in Section 3.1 only considers that all CPU cores are activated without considering the application scalability and power budget. In this section, we consider that an arbitrary number of cores can be activated. Such difference is critical for NUMA architectures and applications that do not scale well. As illustrated in Figure 3.11, we

consider a total of four critical power levels for CPU cores and two critical power levels for memory modules. We note that the exact values of these levels are application-specific but that their relative orders are maintained across all applications.

Critical CPU Power Levels:

- P_{cpu,L_1}: the upper bound of CPU power when all cores are activated and run at the maximum performance state.
- P_{cpu,L_2}: the CPU power when all cores are activated and run at the minimum performance state.
- P_{cpu,L_3}: the CPU power when only one core is activated and runs at the maximum performance state.
- P_{cpu,L_4}: the CPU power when only one core is activated and runs at the minimum performance state.

Critical Memory Power Levels:

- P_{mem,L_1}: the upper bound of memory power when memory runs at the highest speed to support data for all cores.
- P_{mem,L_2}: the memory power when memory supports data for one core running at the maximum frequency.

3.2.2 Application Aware Power Coordination Algorithm

To quickly approximate an optimal solution for the power bounded cross-component power coordination problem, we developed an application-aware hierarchical power coordination method following three techniques:

- Workload classification. We categorized parallel applications based on their parallel scalability and memory access intensity analyses and customized the power coordination strategy for each category. Along each characterization dimension, we labeled each application with one of three values: low, moderate, and high, which we have determined empirically.

- Search space reduction. We reduced the search space of workload execution configurations by using a small set of pivot execution configuration, which is crucial to provide key reference data and has a high probability of being an optimal solution.
- Ordered search. We prioritized the parameters of the execution configurations and used that order to navigate the search process. In other words, if control knob A has a larger performance or power impact than control knob B, we determined the parameter for A first.

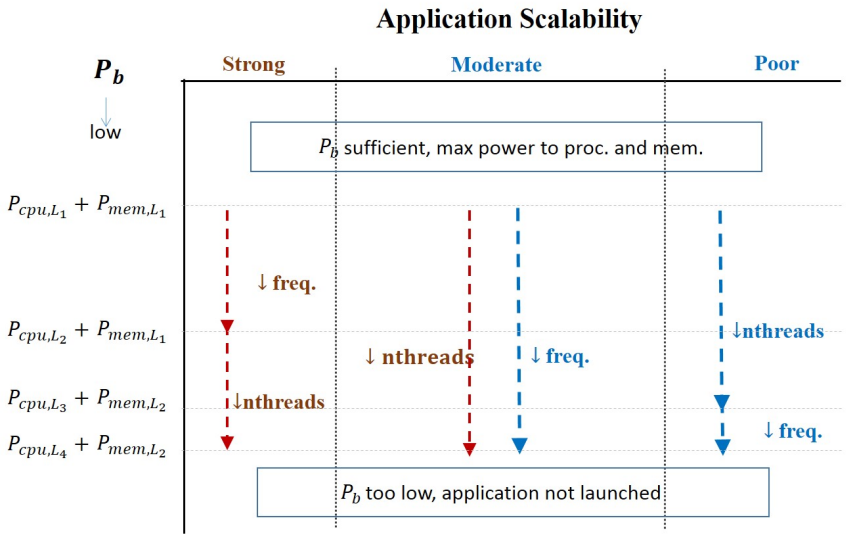


Figure 3.11: Application aware power coordination strategy.

As illustrated in Figure 3.11, the algorithm acts in four steps:

Step 1 decides the power budget level. If $P_b < P_{\mathtt{cpu},L_4} + P_{\mathtt{mem},L_2}$, the algorithm reports "*power budget too low*". If $P_b > P_{\mathtt{cpu},L_1} + P_{\mathtt{mem},L_1}$, the algorithm allocates the maximum power to processor and memory that the application can possibly consume and adapts the thread concurrency and affinity with the highest performance state. In addition, the algorithm also reports to system the excessive power budget that is not needed for the workload. Otherwise, the algorithm proceeds to the following steps.

Step 2 decides processor-memory power distribution and prioritizes the memory power allocation based on the heuristic power coordination algorithm introduced in Section 3.1.

Step 3 decides the number of activated cores and core frequency. The strategy is based on the application scalability. For strongly scalable applications, the algorithm first attempts to

activate as many cores as the power budget allows and then runs the processor cores at the highest speed possible. For poorly scalable applications, the scheduler first attempts to run activated cores at the highest frequency and then tries to activate more cores if the power budget allows. For applications with moderate scalability, the algorithm simultaneously determines the target concurrency and core speed to match the applications' memory intensity and parallel scalability.

Step 4 identifies an optimal core affinity. The algorithm makes the decision to allocate cores to be distributed or consolidated cross NUMA architecture based on application remote/local memory access patterns and the hardware capability.

3.2.3 Experimental Results and Discussion

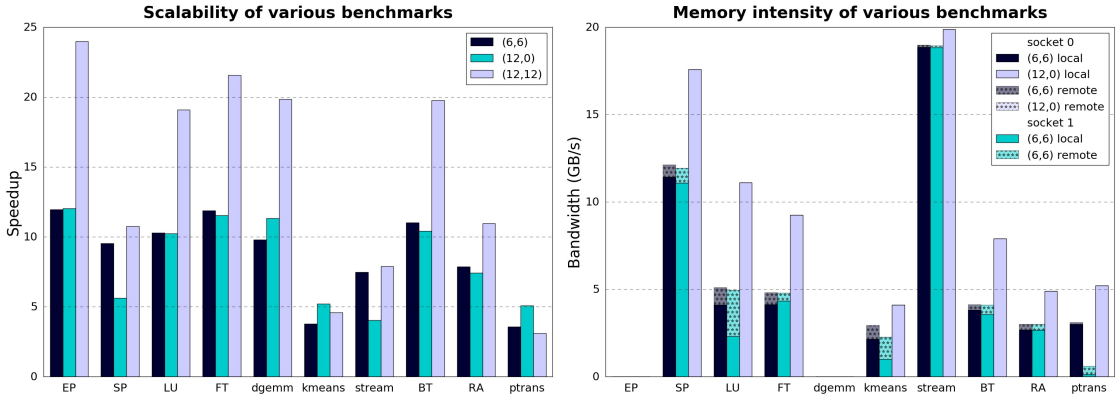
We evaluated our proposed framework and algorithm on one of the 8 Haswell nodes. The details of Haswell processor architecture is described in the Appendix. We measured and capped the power consumption of individual processor packages and DRAMs using RAPL. Table 3.3 lists the parallel benchmarks used for evaluation. These benchmarks have different workload characteristics, spanning from highly scalable, computing intensive to moderately scalable, memory intensive.

Table 3.3: List of benchmarks used in this study

Benchmark	Problem size/Input	Description and Workload Pattern
DGEMM	Ns =32000	Double precision matrix multiplication, compute intensive
Kmeans	numObject=819200	Clustering algorithm, random memory and memory intensive
STREAM	Array size = 8e7	Synthetic benchmark, measuring memory bandwidth
RA	2^{25}	Random memory access
EP	C	Embarrassingly Parallel, compute intensive
Ptrans	N=40000	Parallel matrix transpose, memory intensive
FT	C	Discrete 3D fast Fourier Transform, compute/memory
BT	C	Block Tri-diagonal solver, compute intensive
LU	C	Lower-Upper Gauss-Seidel solver, compute/memory
SP	C	Scalar Penta-diagonal solver, compute/memory

3.2.3.1 Application Characteristics of Evaluated Benchmarks

Parallel Scalability Figure 3.12a visualizes how each benchmark scales with core counts at a fixed core frequency of 2.3 GHz. As illustrated, EP, Dgemm, BT, LU, and FT are highly scalable applications. The speedup almost doubles while increasing the number of cores from 12 to 24 for the



(a) Speedup (b) Bandwidth

Figure 3.12: Parallel speedup and memory bandwidth of different benchmarks under several pivot configurations. The experiment is then evaluated on the Haswell node.

group. SP, RA, and STREAM are moderately scalable applications, whose performance scales well while the number of cores is less than 12, but becomes poor when using more than 12 cores. Kmeans and Ptrans are poorly scalable applications, whose maximum speedup is less than 6 even when using all 24 cores. There are two major reasons that Kmeans and Ptrans perform poor scalability. First, both Kmeans and Ptrans have more remote bandwidth demand than local bandwidth demand, thus it would not improve performance to run threads cross sockets. Second, due to the algorithm design, the application fails to exploit the available hardware scalability all that well.

The profile results lead to a heuristics that the performance data at execution configurations using n, $n/2$, and $n/4$ are adequate to cluster the applications into three categories: `scalable`, `moderately scalable`, and `poorly scalable`. Here n denotes the total number of cores on the system, and the application also runs n threads with each thread is mapping to one single core on the system.

Memory Intensity and Core Affinity Memory intensive applications transfer data heavily from/to local and remote memory. As shown in Figure 3.12b, memory intensive applications like Kmeans and SP achieve more than 10 GB/s with half of the available cores, while computer intensive applications like EP and Dgemm almost require no memory access.

Further, we can split memory intensive applications into two subcategories based on the remote/local memory access ratio. Applications in the first subcategory involves large amounts of local memory accesses but minimal remote accesses. This is because the local memory bandwidth of

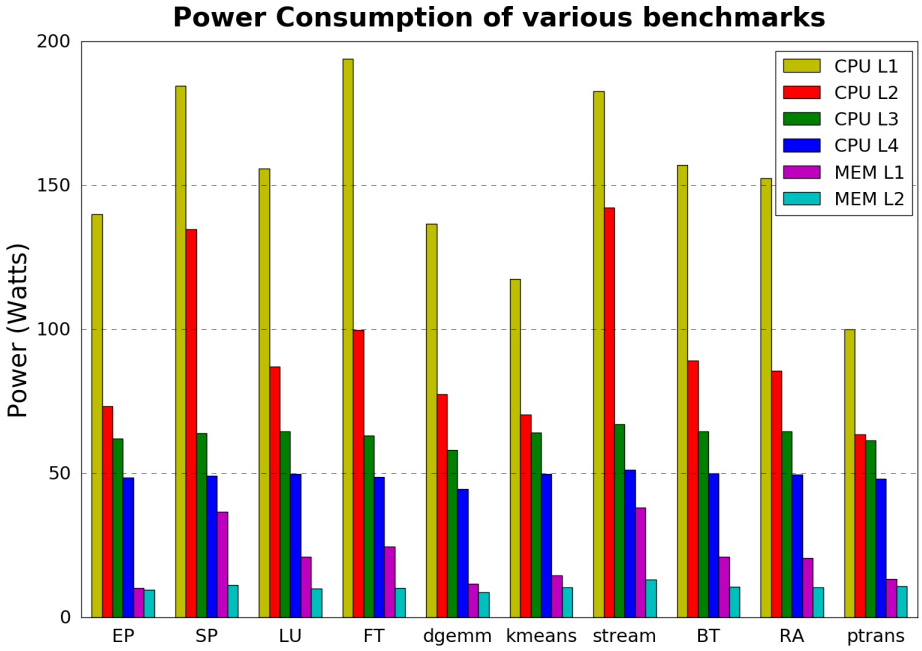


Figure 3.13: Applications' critical power levels.

hardware could be saturated and cause memory contention which significantly degrades performance. In such case, distributing cores to more sockets increases the application performance. SP and STREAM are typical examples that belong to the first subcategory. Applications like Kmeans and Ptrans can be classified to second subcategory, as they have large ratios of remote to local memory accesses when executing across multiple sockets. For applications that fall into the second subcategory, consolidating cores to a same processor improves both performance and power efficiency.

Workload Critical Power Levels Figure 3.13 summarizes the applications' power and performance under critical power levels, which validates our rationales of algorithm design and demonstrates the usefulness of pivot configurations. Figure 3.13 shows the application's critical power level measured on the Haswell Dual-processor node. Particularly, the power levels could vary between different architectures for the same application.

3.2.3.2 Evaluation of Application Aware Power Allocation Strategies

To evaluate how well the proposed scheduler (**proposed**) behaves, we compare it with five other coordination methods under various given power budgets.

- **Optimal**, which uses exhaustive search to identify a power allocation which delivers the best

performance for a given power budget.

- **All-core**, which optimally distributes power budgets between processors and memory using all cores by adopting heuristic power coordination introduced in Section 3.1 [42].
- **Consolidated**, which uses the same strategy as ours to identify cross processor-memory power allocation and the right number of cores, but always places the activated cores on the same sockets when possible.
- **Distributed**, which is similar to consolidated, but always try to evenly place the activated cores among the sockets.
- **Max-mem**, which always allocates maximum memory power and leaves the rest power budget to processors. On the experimental system, the maximum memory power allocation is 40 Watts obtained from application STREAM.

We evaluated the proposed methodology on the Haswell Dual-processor node with Turbo option disabled. The detailed information about the experimental node is presented in Appendix. Figure 3.14 summarizes the comparison of the six methods. Note that the 240 Watts power budget approximates an unbounded performance. We use parallel speedup as the performance metric, which is calculated using the performance of sequential execution at the highest core performance state as the base case. This figure supports the following observations.

- *Proposed* performs similar to *Optimal*, leading to either same solution as *Optimal* or solution close to *Optimal*. On average, the performance difference is within a 4% range for all experiments. Particularly, configurations returned by *Proposed* produce best performance among the six methods when the power budget is sufficient.
- *Proposed* outperforms *Max-mem* for almost all cases and doubles performance for certain cases, supporting the argument that cross-component power allocation must be application- and budget-aware.
- *Proposed* outperforms *All-core* when the application is not strongly scalable or the given power budget is constrained and delivers doubled performance over our experiments.
- *Proposed* outperforms *Consolidated* or *Distributed* for memory intensive applications, supporting that core affinity is a useful technique for power bounded computing.

To conclude, the experimental results confirm that the proposed coordination is efficient and effective to address the problem of power bounded computing on NUMA multicore systems.

3.3 Summary

As HPC systems are increasingly bounded by power budget, it is clear that future HPC systems and softwares must cope with these power bounds. In this chapter, we studied the coordinated power allocation between processors and memory modules and presented a novel power-bounded computing framework for modern NUMA-enabled multicore systems. This framework explores multidimensional power management techniques to simultaneously maximize performance and enforce a power budget. This framework includes an application-aware cross-component power coordination method and a prototype implementation. Overall, this work makes the following contributions:

- We present a practical application-aware power coordination method and implementation. This method quickly determines the hardware and power resource allocation that delivers (near) optimal performance for applications on NUMA multicore systems.

- We show that application characteristics including power and performance scalability and memory access intensity are indicative for guiding cross-component power coordination on power bounded systems.

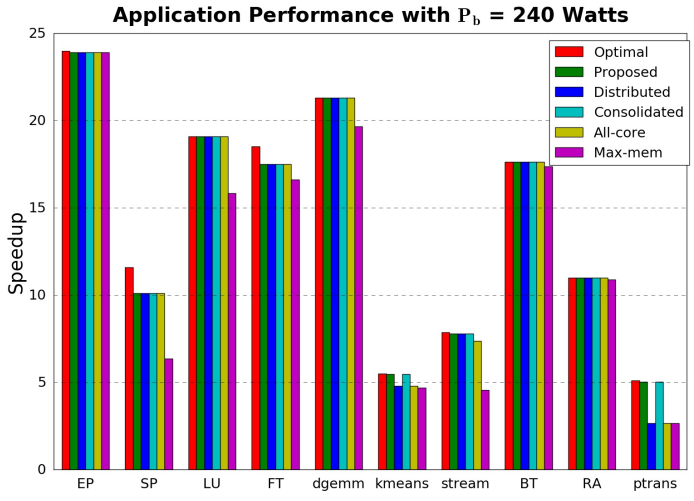
- We show that using multiple software and hardware mechanisms such as coordinated concurrency, core/memory affinity, and component power level limiting can considerably improve performance under the same power budget. We demonstrate that core affinity has a considerable power and performance impact on power bounded computing on NUMA architectures.

- We show impacts of cross-component power coordination. In a power-bounded system, the allocation of the power budget among competing components significantly affects application performance. We discover that there exist six categories, each of them summarizing a distinct pattern of how application performance and actual power respond to different power allocations. Further, we explain how the underlying power capping framework attributes to this categorization.

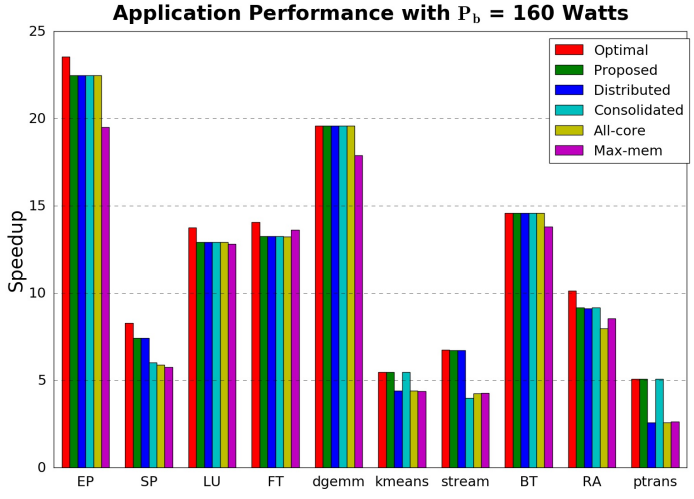
- We find a clear linkage between processor and memory' critical power levels and categories of

power allocation scenarios. This linkage also indicates the proper range of power budget to deliver desirable performance and power efficiency.

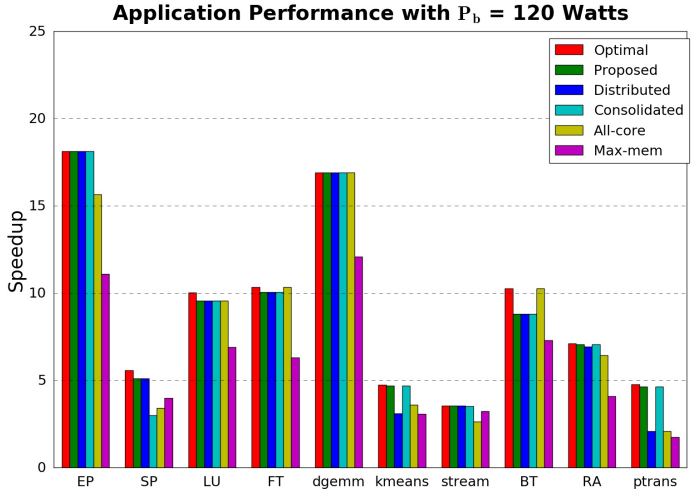
We envision that as power becomes a scarce resource, power bounded approach presents a new perspective to address the power challenge in HPC. By formulating and solving a node-level power coordination problem, we can extend the work to enable power bounded computing at large scale. In the next chapter, we will discuss the details of extending the work in a cluster.



(a) $P_b = 240$ Watts



(b) $P_b = 160$ Watts



(c) $P_b = 120$ Watts]

Figure 3.14: Performance comparison of different power allocation methods.

Chapter 4

Cluster Level Intelligent Power Coordination

To increase HPC performance, HPC systems resources are rapidly increasing in multiple dimensions including the number of nodes, the number of cores per processor, and the size of shared memory. While today's large-scale systems provides intensively parallelism scalability, workloads can not use the underlying hardware at full scale due to algorithm design, workload partition, data movement and communication. Furthermore, power capping causes performance degradation when the power budget is insufficient to support the concurrency configuration.

In a cluster, optimally managing power for HPC workloads requires an intelligent strategy to control the number of participating nodes and to allocate the available power budget to different subsystems (CPU-core, CPU-uncore and memory) within nodes. The cluster level offers more space to increase system performance under a power bound but at the same time brings new challenges. Inappropriately assigned nodes can either cause inefficient utilization of the available power or can lead to subsystems running at ineffective power levels, thereby delivering inferior performance. If fewer nodes are assigned, each activated node gets excessive power budget than demand and loss the potential higher applications' parallelism at the cluster level. Contrarily, if more nodes are assigned, each node receives insufficient power. Correspondingly, the power states of components on each node are not operating at their optimal states, which leads to significant performance degradation. Managing power on the cluster level requires balance between the cluster, node, and component

levels to avoid power waste and performance degradation.

In this chapter, we investigate power-bounded computing on multicore-based systems and develop the Cluster Level Intelligent Power (CLIP) coordination framework. CLIP employs application-aware power bounded scheduling for parallel applications on clusters built of NUMA multicore nodes. It characterizes the scalability of parallel applications and their power demands, and accordingly recommends the optimal application execution configuration and power distribution. The framework implementation is hierarchical and consists of two levels: the cluster level determines the number of nodes and the power budget for each node; the node level selectively activates the CPU cores and distributes the available power budget to the CPU and memory within nodes. The framework uses light-weight off-line profiling for application characterization, and classifies workloads into three categories. It delivers desirable performance and meets the power budget with four steps:

- Identify the number of participating nodes based on application scalability and power demand on each node.
- Allocate a per-node power budget between CPU and DRAM based on the application's features.
- Choose core and memory affinity based on application memory access intensity.
- Identify the optimal number of active cores based on the application's scalability at the node level.

In the following sections, we will discuss the detail of CLIP. Section 4.1 discusses how the system determine a configuration on node level according to application characteristics. Section 4.2 discusses the power allocation strategy on Cluster Level. Section 4.3 presents the framework design and Section 4.4 evaluates CLIP and compares with other power coordination approaches. We conclude a summary in Section 4.5.

4.1 Configuration Selection on Node Level

With knowledge of the correlation between performance and configuration under multiple power budgets, the cluster can determine the number of participating nodes to ensure that the node is running on reasonable power states. In Chapter 3, we discuss how to adjust number of cores and

map core affinity across NUMA architectures, and further power coordination among main computer components (CPU and DRAM). Here, we investigate deeper about the performance scalability under multiple components performance states. Based on these knowledge, the system can provide more accurate performance and power estimation to support cluster level management.

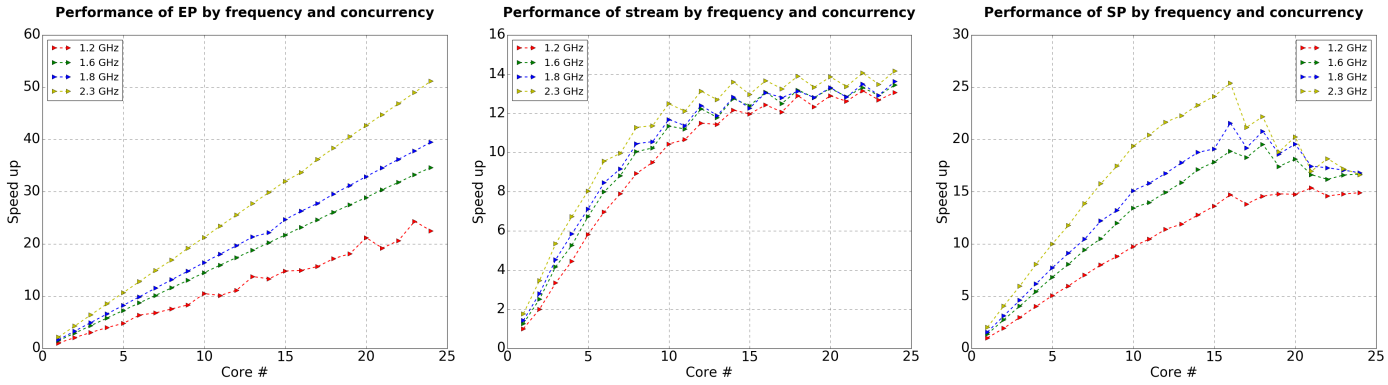
4.1.1 Scalability Trend

Scalability describes how application performance changes on parallel computing systems. $S(n) = \frac{\text{perf}(n)}{\text{perf}(1)}$ describes the speedup with the number of utilized cores n. Similarly, scalability can describe how application performance changes with processor's speed. We define $S(\text{freq}) = \frac{\text{perf}(\text{freq})}{\text{perf}(f_{\text{lowest}})}$. Here, freq is the processor frequency. $S(\text{freq}) \propto \text{freq}$ holds for most applications, while $S(n) \propto n$ is true only for ideal or embarrassingly parallel applications.

To observe the impact of number of involved cores and core frequency to application performance and power, we run each application with with threads number from 1 to 24. In each configuration, we only execute one thread per core. The application input parameters are listed in Table 4.2.

Figure 4.1 illustrates how applications perform with various processor frequency and processor count on one of the Haswell nodes. As seen from Figure 4.1, there are three types of scalability trends on parallel architectures, which we denote as *linear*, *logarithmic*, and *parabolic*. Even though the three trends are not able to describe application performance comprehensively, they are able to cover most typical HPC parallel applications. The performance of *linear* applications increases linearly with concurrency and processor frequency. The performance of *logarithmic* applications increases linearly until an inflection point, after which performance growth drops. The performance of *parabolic* applications increases linearly when concurrency is less than the global maximum. Beyond the global maximum, increasing concurrency causes performance degradation. We observe that there is an inflection point in each of the non-linear curves in Figures 4.1b and 4.1c, which separates the scalability trends into two segments. Both *logarithmic* and *parabolic* can be approximated by a piecewise model, i.e., a linear piecewise segment ($S(n) \propto n \, (n \leq NP) \mid NP$ is the inflection point), and the remaining segment.

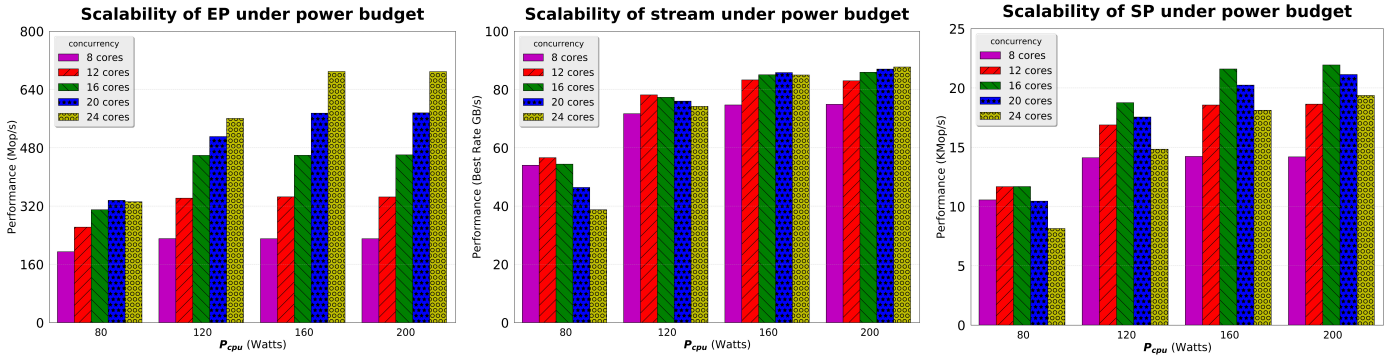
Figure 4.2 shows how a power budget would impact the three types of applications differently. For a *linear* application like EP in 4.2a, the performance is best at highest concurrency unless power is lower than the lower bound of the acceptable power. For *logarithmic* applications, the number of



(a) linear (b) logarithmic (c) parabolic

Figure 4.1: Scalability trends of *linear* (4.1a), *logarithmic* (4.1b), and *parabolic* (4.1c) applications.

cores activated to achieve best performance decreases with the power budget, as shown in Figure 4.2b. For *parabolic* applications, the insufficient power budget exacerbates the performance loss of all-core configuration as seen in Figure 4.2c. The performance gap between the optimal concurrency and maximum concurrency also increases with the power budget decreasing.



(a) linear (b) logarithmic (c) parabolic

Figure 4.2: Performance impact of processor power budget for *linear* (4.2a), *logarithmic* (4.2b), and *parabolic* (4.2c) applications.

To classify the application scalability trend, we simply compare the performance under two profiling stages: Perf_{all} and $\text{Perf}_{\text{half}}$ to determine the application scalability types. Perf_{all} and $\text{Perf}_{\text{half}}$ denote the performance with all and half of the available cores respectively. The applications with $\frac{\text{Perf}_{\text{half}}}{\text{Perf}_{\text{all}}} < 0.7$ are classified as *linear* type; the applications with $0.7 \leq \frac{\text{Perf}_{\text{half}}}{\text{Perf}_{\text{all}}} < 1$ are classified as *logarithmic* type; and applications with $\frac{\text{Perf}_{\text{half}}}{\text{Perf}_{\text{all}}} \geq 1$ are classified as *parabolic* type.

4.1.2 Performance Prediction Model

Linear Type The execution time of *linear* type applications can be modeled with a linear function:

$$Time_t = \sum_{i=1}^{m} (Time_i \cdot \alpha_{(t,i)}) + \lambda_t \tag{4.1}$$

The terms $\alpha_{(t,i)}$ and λ_t describe the relationship between the target configuration run time and the sample configuration run time. m denotes the number of sample configurations in the profile stage. For the *linear* type, we only need to profile half-core and all-core sample configurations to implement the equation. $\alpha_{(t,i)}$ scales up the recorded execution time, $Time_i$, on the sample configurations and reflects the scalability based on hardware event rates.

Logarithmic Type Seen from Figure 4.1b, the speedup of the workload increases linearly while the thread number is less than or equal to a specific number NP, the inflection point of the function. When the thread number is larger than NP, the performance trend can also be approximated with another linear model with a much smaller slop to fit the trend. Therefore, we conclude a linearly piecewise function to describe the trend of *logarithmic* applications without impacting the accuracy significantly.

The key to estimate an accurate piecewise model is to predict the inflection point NP accurately. We use multivariate linear regression (MLR) to identify the infection point because its relative low requirement of amount of data and acceptable accuracy. The MLR model utilizes the event rates and the manually identified inflection point NP for model training. The involved events is listed in Table 4.1.

Table 4.1: The Haswell hardware events used in sample configurations for prediction.

Predictor	Description
Event0	Instruction Cache (ICACHE) Misses
Event1	Memory Access Read Bandwidth
Event2	Memory Access Write Bandwidth
Event3	L3 Cache Miss from Local DRAM
Event4	L3 Cache Miss from Remote DRAM
Event5	Cycles Active
Event6	Instructions Retired
Event7	Performance ratio by full cores and half cores

After obtaining the inflection point NP, the two segment linear functions can be derived

as:

$$\begin{cases} Time_t = \sum_{i=1}^{m}(Time_i \cdot \alpha_{(t,i)}) + \lambda_t & if\ \ t \leq NP \\ Time_t = \sum_{i=1}^{m'}(Time_i \cdot \alpha'_{(t,i)}) + \lambda'_t & if\ \ t > NP \end{cases} \tag{4.2}$$

Equation 4.2 illustrates two slope and intercept parameters to represent the scalability growth difference. Since power increases close to linearly with the participating cores count [44], it is not sufficient to run the application with concurrency in the latter segment while power is not sufficient to keep all running cores at the highest frequency. As seen in Figure 4.1b, frequency significantly impacts the application's performance. Therefore, we would prefer high frequency to high concurrency for *logarithmic* applications. Thus, this model offers support for efficiently exploiting cluster-level power allocation on each node.

Parabolic Type. For *parabolic* applications, we use MLR model to predict the inflection point NP. As more participating cores could generate poorer performance and higher power consumption, we only estimate the function for the former segment as:

$$Time_t = \sum_{i=1}^{m}(Time_i \cdot \alpha_{(t,i)}) + \lambda_t \qquad if\ \ t \leq NP \tag{4.3}$$

Since the latter segment consumes more power and obtains lower performance, we disregard the prediction for the $n > NP$ segment.

The relation of the application performance variation under different power budget and concurrency is essential to determine node level concurrency configuration and power allocation. Further, it offers support for node level power tuning. The system identifies the reasonable power budget on the node level and decides the number of nodes in a cluster for a given cluster power budget.

4.2 Power Allocation on Cluster Level

Power coordination at the cluster level needs to consider: (1), how many nodes should be involved in the computation for current workload; (2), how to allocate power on each activated node. To achieve the maximum performance, we first use the prediction model to obtain the optimal number of threads in node level; then we could speculate the power consumption of the CPU and memory on the node with different frequencies according to the power model. With the available power range

on each node, the system determines the number of nodes by predicting the performance with different configurations for the given cluster power budget. Lastly, we utilize the cross-component power coordination to deliver optimal node performance as described in Chapter 3.

In Chapter 3, we identify that each application has its specific power levels for CPU and memory. $P_{\texttt{cpu},L_1}, P_{\texttt{mem},L_1}$ are the CPU and memory power consumption when the CPU runs at the highest CPU frequency. The reasonable power range of an application to run efficiently falls into $[P_{\texttt{cpu},L_2} + P_{\texttt{mem},L_2}, P_{\texttt{cpu},L_1} + P_{\texttt{mem},L_1}]$. The power for each component on a node can be derived from the following equations. First, the total cluster power budget can be distributed as:

$$P(job) = P_1 + P_2 + ... + P_k + ... + P_n \tag{4.4}$$

The power P_k on each node is decomposed as the aggregated sum of components types, including processors, memory, and other:

$$P_k = P_{ProcT} + P_{MemT} + P_{OtherT} \tag{4.5}$$

The processors' power P_{ProcT} can be formulated as the sum of individual processors $P_{proc,i}$.

$$P_{ProcT} = \sum_{i=1}^{NS} P_{proc,i} \tag{4.6}$$

NS dotes the number of sockets (processors) here. Each processor's power can be further divided as a base power $P_{base,i}$ and the cores' power, which is different while running on different workloads w.

$$P_{proc,i} = P_{base,i} + \sum_{j=1}^{NC} P_{c_j}(w) \tag{4.7}$$

Similarly, the power of memory P_{MemT} can be typed as the sum of memory components' power which is broken down with a base power $P_{m_{base},i}$ and an activity power $P_{m_{load},i}(w)$:

$$P_{MemT} = \sum_{i=1}^{NS} P_{mem,i} \tag{4.8}$$

$$P_{mem,i} = P_{m_{base},i} + P_{m_{load},i}(w) \tag{4.9}$$

Manufacture Variability Manufacture variability could cause performance loss across nodes in a large scale system. Inadomi's work [52] demonstrated that manufacture variability increases significant imbalance among nodes and rises the synchronization cost. The cluster should estimate the float power demands by each node and keep the core frequency across node to be as close as possible to alleviate the imbalance caused by manufacture variability. In our experimental environment, the variability between the eight Haswell nodes is ignoble, thus we do not consider manufacture variability in the cluster power coordination.

4.3 System Design

Figure 4.3 shows the framework of CLIP. CLIP includes a profiling module, a data-driven execution configuration recommendation module, an application execution module, and several helper tools to provide an user-friendly power-bounded computing environment.

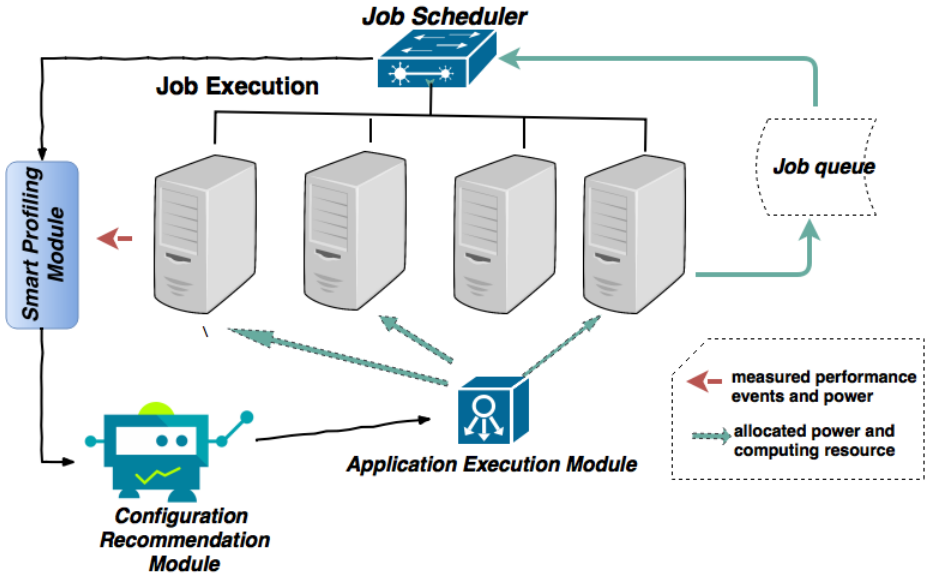


Figure 4.3: Overview of CLIP

4.3.1 CLIP Components

1. *Profiling Module*: The profiling module runs several iterations of the application's kernel function with sufficient power. The system collected performance events and execution time information for future affinity determination and scalability trend classification.

2. *Configuration Recommendation Module*: The configuration recommendation module takes the profiling data and power budget as inputs and return a parallel workload execution configuration.

3. *Application Execution Module*: The application execution module first checks whether the database contain the profiling data of the workload. If the response is negative, it will require the profiling module to profile the workloads first and input the profiles data to the recommendation module to get the suggested configuration. The application execution module submit the jobs with the suggested configuration to the power-bounded multicore cluster.

4. *System Interface Helper Tools*: The module includes several customized system tools such as a power meter reader, a performance state controller, a power capping controller, and a performance event collector.

4.3.2 Cluster Power-Bounded Scheduling Algorithm

As illustrated in Algorithm 2, the proposed scheduler acts in two steps:

1. The scheduler searches for the given job in the knowledge database to decide if it is necessary to start smart profiling. By smart profiling or searching from the knowledge database, the system is able to acquire the optimal power range $[P_{\texttt{cpu}_{L_o}} + P_{\texttt{mem}_{L_o}}, P_{\texttt{cpu}_{H_o}} + P_{\texttt{mem}_{H_o}}]$ for each node. After that, the system inputs the profile data and the given power budget recommendation to decide the number of nodes and the power budget for each node.

2. The scheduler inputs the power budget for each node and the profile data for each applications to the recommendation module and gets the suggested power budget for the CPU and memory, the number of activated cores, and the optimal core affinity.

4.4 Experimental Results and Discussions

To evaluate the performance of CLIP, we compare it with three other coordination methods under various given power budgets. The benchmarks for evaluation are listed in Table 4.2. The experimental platform is an 8-node Haswell Cluster. The details of cluster are described in Appendix.

Algorithm 2 CLIP (Cluster level intelligent power coordination system).

function CLIP(*App*, *C*)

Input: P_{ub}: the total power budget for the cluster;
Input: *App*: the application under study
Input: *C*: the cluster with N_{total} nodes
Input: N_{total}: the total number of nodes in the cluster *C*
Output: N_{nodes}: suggested number of active compute nodes
Output: $P_{\texttt{cpu}_{run_i}}$: suggested CPU power for node *i*
Output: $P_{\texttt{mem}_{run_i}}$: suggested memory power for node *i*
Output: N_{cores}: suggested number of active cores on each node
Output: *Map*: suggested mapping affinity

$[P_{\texttt{cpu}_{H_o}}, P_{\texttt{cpu}_{L_o}}, P_{\texttt{mem}_{H_o}}, P_{\texttt{mem}_{L_o}}, \text{Profile}] \leftarrow SmartProf(App)$

$[N_{core}, Map] \leftarrow$ Recommendation (Profile)

if *App* has a set of predefined number of processes $N_{def_1}, ..., N_{def_n}$ **then**
  if $N_{def_k} \leq P_{ub}/(P_{\texttt{cpu}_{L_o}} + P_{\texttt{mem}_{L_o}}) < N_{def_{k+1}}$ **then**
    $N_{nodes} \leftarrow N_{def_k}$
    $P_{node} \leftarrow P_{ub}/N_{nodes}$
    for every node *i* to be activated **do**
      $[P_{\texttt{cpu}_{run_i}}, P_{\texttt{mem}_{run_i}}] \leftarrow P_{node}$ (Equation 4.5)
    end for
  end if
else
  if $P_{ub} > N_{total} * (P_{\texttt{cpu}_{H_o}} + P_{\texttt{mem}_{H_o}})$ **then**
    $N_{nodes} \leftarrow N_{total}$
  else
    $N_{nodes} \leftarrow P_{ub}/(P_{\texttt{cpu}_{H_o}} + P_{\texttt{mem}_{H_o}})$
  end if
  for every node *i* to be activated **do**
    $P_{\texttt{cpu}_{run_i}} \leftarrow P_{\texttt{cpu}_{H_o}} + P_{\texttt{cpu}_{v_i}}$
    $P_{\texttt{mem}_{run_i}} \leftarrow P_{\texttt{mem}_{H_o}} + P_{\texttt{mem}_{v_i}}$
  end for
end if
end function

Table 4.2: List of benchmarks used for cluster power coordination

Benchmark	Description	Parameters	Workload Pattern	Scalability Type
BT-MZ	Block Tri-diagonal solver	C	compute	logarithmic
LU-MZ	Lower-Upper Gauss-Seidel solver	C	compute/memory	logarithmic
SP-MZ	Scalar Penta-diagonal solver	C	compute/memory	parabolic
CoMD	classical molecular dynamics	-n 240 240 240	compute	linear
AMG	algebraic multigrid solver	-n 300 300 300	compute/memory	linear
miniAero	mini version to solve the compressible Navier-Stokes equations	default	compute	parabolic
miniMD	force computations	default	compute	linear
TeaLeaf	solves the linear heat conduction equation	Tea10.in	compute/memory	parabolic
CloverLeaf	solves the compressible Euler equations on a Cartesian grid	clover128_short.in	computer/memory	logarithmic
CloverLeaf	solves the compressible Euler equations on a Cartesian grid	clover16.in	computer/memory	logarithmic

All-In. This utilizes all supplied nodes. It allocates 30 watts to memory and the remaining power to CPU on each node without considering the cluster power budget. All of the cores participate in application execution. To be specific, allocating 30 watts to memory meets most applications'

memory power requirement and won't cause very significant degradation for extremely memory intensive applications' performance.

Lower Limit. This method ensures that no nodes participating in the computation are allocated a budget less than a preset value, i.e., 180 Watts. If the total power budget cannot allocate every node more than 180 watts, the scheduler decreases the number of active nodes. Additionally, this method utilizes all cores on each active node and allocates 30 watts to memory.

Coordinated [42] as described in Chapter 3. This method ensures that the nodes participating in computation are allocated a budget no less than a preset value specific to the application [42]. It coordinates power between CPU and memory according to the power model. The **Coordinated** method executes applications at the highest possible concurrency.

CLIP. It guarantees that the participating nodes are allocated with a budget no less than the lower bound of the acceptable power range for the specified application. Therefore, it decreases the node count to ensure each node has a reasonable power budget if the cluster power budget is not sufficient for all supplied nodes. Besides, **CLIP** changes the concurrency on each node according to the application scalability type and total power budget, and also coordinates power allocation between CPU and memory.

Figures 4.4 and 4.5 summarize our comparison of the four methods. In the comparison, we use the relative performance based on the **All-In** method without a power bound. The two figures support the following observations:

1. **CLIP** achieves similar performance as **All-In** for most of the applications under study, and outperforms $\geq 40\%$ for MiniMD and SP-MZ applications of the *parabolic* type, when there is no specified power bound.

2. **CLIP** performs best for all the tested benchmarks if the power budget is unlimited or high.

3. **CLIP** outperforms **All-In**, **Coordinated**, **Low-Limit** for most cases, specially for *logarithmic* and *parabolic* applications.

4. **CLIP** defeats **Coordinated** for *parabolic* applications (SP-MZ, miniAero and TeaLeaf) by up to 60% overall. When the thread count exceeds optimal, these *parabolic* applications experience

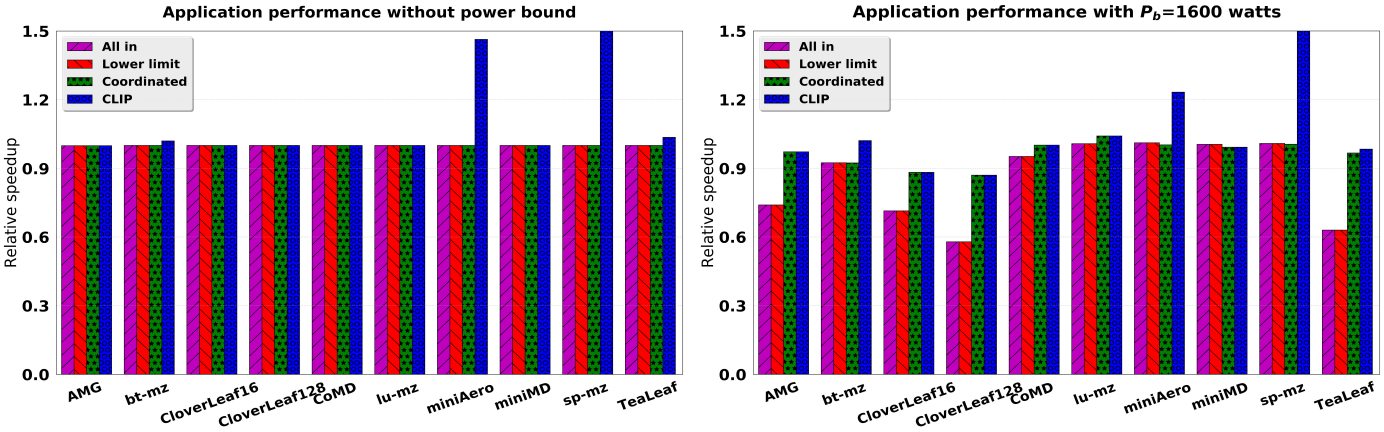


Figure 4.4: Performance comparison of different power allocation methods under high power budgets

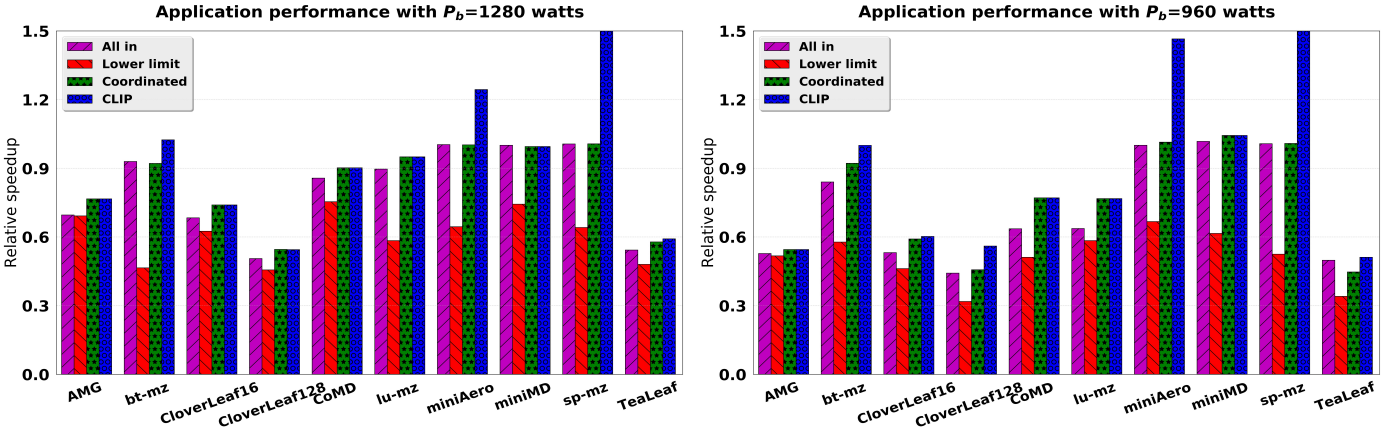


Figure 4.5: Performance comparison of different power allocation methods under low power budgets

a worsened performance but consume more power. Carefully distributing resources for such applications significantly improves performance.

5. **CLIP** outperforms **Coordinated** for *logarithmic* when the power budget is low. *logarithmic* applications is common among big data applications that require higher memory bandwidth. This observation confirms the hypothesis that classifying applications and setting corresponding configurations is beneficial for power-bounded computing.

4.5 Summary

In this chapter, we present CLIP, a framework for cluster-level power-bounded resource coordination on NUMA multicore based systems. The power coordination methodology on Cluster Level (CLIP) achieves better performance, as it considers both cluster-level and node-level power coordination and resource allocation on NUMA multicore-based systems. Overall, the major findings and contributions of this work include:

- We show that power-aware hardware and workload execution management improves both performance and power efficiency for power-constrained systems.
- We propose an application-aware power coordination method, which comprises application characterization and performance modeling. This method can identify a (near) optimal configuration without exhaustively searching the configuration space.
- We implement the power coordination framework and evaluate it on real systems with multiple applications. Experimental results show the framework performs much better than the default or only node level power coordination methods under various power budgets.
- We present several findings and insights on concurrency configurations for high performance power-bounded computing.

We admit that CLIP has some limitations, like it requires profiling and will coordinate the cores count and the nodes count which may not fit for some applications. However, CLIP reveals some key insights about power bounded computing on large scale systems and provides a solution with a low overhead. The idea to tune the system on both cluster and node levels can be applied to large scale systems.

Chapter 5

Contention Aware Power Bounded Scheduling

HPC resources management has transformed from a course-grained way to a fine-grained approach to improve HPC system utilization. In a lot of middle size supercomputers, like Palmetto [1], Ohio Supercomputer Center, etc shares a single node between jobs. Such node sharing mechanisms allow users to spend their assigned resource hours more finely and efficiently, and further improving system hardware utilization.

While resource sharing has been investigated a lot, job scheduling considering both power and hardware resource is a fundamentally new problem. In previous research, contentions arise when workloads compete for shared hardware resources like memory bandwidth, last level cache, and interconnection. Prior job collocation strategies are problematic when power is limited on systems, nodes, and components. Power constrains the amount of activated hardware resources and their capacity for workload execution, thus inducing or aggravating contention, particularly on the memory hierarchy. Furthermore, when the total power is limited, balancing power among nodes and components is critical. Under-provisioning on some components can lead to severe contention and performance degradation. Understanding the impact of power allocation at the system, node, and component levels is necessary to mitigate contention and performance interference.

In this chapter, we study how the power limiting affect of contention among collocated scientific parallel jobs in multicore based clusters, and research effective strategies to mitigate contention

and maximize system performance under given power budgets. This work is timely, as modern server nodes comprise multiple processor cores and many data centers begin to adopt node sharing mechanism [93, 4].

We use machine learning methods to predict contention using application performance and power profiles, and present *CAPS*, a Contention-Aware Power-bounded Scheduling that mitigates contention and coordinates power between nodes and components. Overall, *CAPS* embraces two key ideas: (1) infer the contention using applications' performance and power profiles and its variation with power limits, (2) exploit job collocation *and* supportive power distribution across nodes and components to mitigate contention caused by power limits.

In Section 5.1, we discuss the benefits and challenges of power-bounded job and resource co-scheduling. Section 5.2 presents the methodology and implementation of CAPS. The experimental results are evaluated in Section 5.4. Finally, we summarize and conclude the findings in Section 5.5.

5.1 Benefits and Challenge of Resource Sharing

5.1.1 The Benefits of Resource Sharing

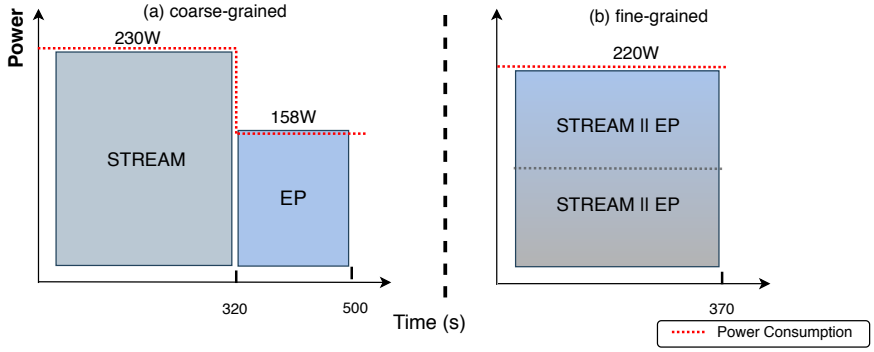


Figure 5.1: Throughput comparison between course grained and fine grained (collocation) resource scheduling.

Figure 5.1 shows the throughput difference between two scheduling methods. We run STREAM and EP on an Intel Haswell Dual-processors node. The specific application parameters are presented in Table 5.3. The coarse-grained method runs STREAM with 24 threads and EP with 24 processes serially. The fine-grained method execute STREAM and EP concurrently

with each workload occupy half core resources. Correspondingly, the threads/processes number of STREAM/EP is declined to 12 for each application.

As shown in Figure 5.1, sharing a node between EP and STREAM improves the utilization of both hardware resources and power budget at the node level. While running STREAM and EP one after another, the power consumption varies from 230 watts to 160 watts without a power cap. The same power budget (e.g. 220 watts) will impact STREAM's performance significantly and is underutilized by EP. A fine-grained scheduling keeps power consumption more stable. More stable power consumption alleviates the burden to the management system power consumption and heat dissipation.

Comparing the performance of the two scheduling approach on STREAM and EP, fine-grained resource scheduling improves the system throughput more than 20%. As STREAM is a memory intensive application, the system memory bandwidth is saturated with only 12 STREAM threads. Increasing the number of threads from 12 to 24 barely increase STREAM's performance, but does increase power consumption significantly. Fine-grained scheduling shares half of the core resources to EP, which cause little interference to STREAM. Therefore, fine-grained scheduling improves system throughput significantly. Memory intensive applications exist widely among HPC applications and are not able to fully exploit the node level core parallel scalability. Allocating just adequate hardware resources for a given parallel job would be better for system efficiency.

5.1.2 The Challenge of Power Bounded Resource Sharing

We have shown that when the node power budget is 220 Watts, it is beneficial to collocate STREAM and EP. However, to make sure resource sharing under a power bound will benefit the system throughput, we face several questions. First, how should the system determine complementary workloads that benefit from resource sharing under a power bound? Second, will job collocation still be beneficial under different power budgets? Third, how will the system allocate power to nodes and components for co-scheduling systems? In this subsection we demonstrate how power limits can change and devoid the benefit of job collocation and motivate the need for proper power distribution.

Limited available power can devoid the benefits of job collocation and make it detrimental for system performance. Take the same jobs STREAM and EP, we repeat the same experiments when the node power budget is 100 Watts. As shown in Figure 5.2, collocating them worsens the system throughput by 50% than when executed sequentially. The limited power budget decreases the core's

computing capacity, causing significant contention between the jobs. Consequently, EP's execution time with collocation almost quadruples, instead of doubles if contention-free when compared to the solo execution if all the cores are given the same power budget.

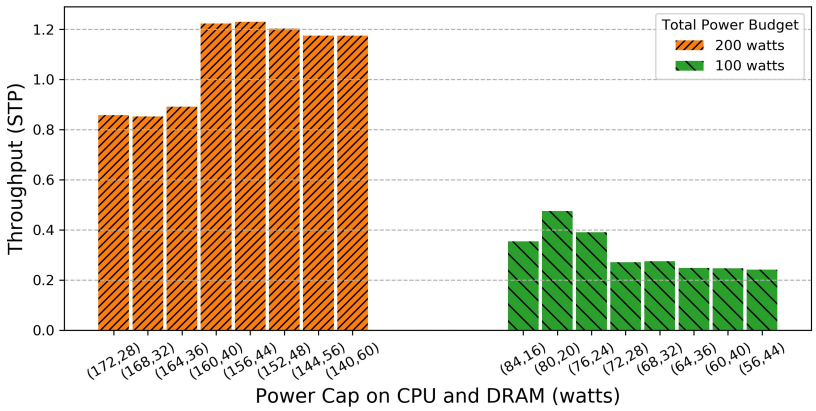


Figure 5.2: The relative system throughput when collocating EP & STREAM under different power budgets and (CPU, MEM) power coordinations.

Furthermore, system performance varies significantly with how the node power budget is distributed to its CPUs and memory. To illustrate the impact of power coordination on a node sharing system, we fix the node power budget at 200 Watts, and repeat the experiments with various CPUs and memory power. As Figure 5.2 shows, there exists a distribution, (156:44) Watts for (CPU: memory), that achieves the highest system performance. Apart from this distribution, shifting power allocation in either direction decreases performance, but shifting power from memory causes larger performance loss.

We can draw three insights from the above discussions. First, job collocation is a practical technique to increase system performance under certain power limits. Second, the proper scheduler must be power-aware because power limits can change jobs from non-interfering to interfering. Third, the effective scheduler must adaptively distribute available power to computer components based on the available power and the jobs under study. We take these insights when designing CAPS.

5.2 Methodology

In this section we present the methodology our scheduler uses to determine whether it should collocate jobs and how it should distribute the power budget to support job collocation. The methodology combines machine learning models and heuristic decisions drawn from empirical

observations.

In a power limited cluster, contention between collocated jobs have two sources: (1) shortage of hardware capacity if power is abundant ($P_b = P_\infty$) and (2) additional hardware capacity reduction due to power limits. These contentions affect system throughput (STP) [37], which is defined as

$$STP(P_b) = \sum_{i,j} \left(\frac{T_i^{|}(P_b)}{T_i^{||}(P_b)} + \frac{T_j^{|}(P_b)}{T_j^{||}(P_b)} \right) \tag{5.1}$$

Here $T^{|}$ and $T^{||}$ are the execution time of jobs i and j when they sequentially and concurrently run respectively under the same power budget P_b. Both run use all the CPU cores. STP is a relative metric, and $STP > 1$ indicates collocating jobs i and j gains throughput over sequentially executing them.

5.2.1 Workload Co-run Throughput Prediction without Power Limit

Contention under abundant power P_∞ has been extensively studied on multicore systems [10, 97]. Prior works commonly use hardware performance monitoring counter (PMC) to infer performance loss of each job and the resulting system throughput. A variety of inference methods have been suggested including statistical modeling and linear regression [56]. Recently, neural networks [74] show promising performance loss prediction on modern multicore systems.

In this work, when power is abundant ($P_b = P_\infty$), we adopt a similar neural networks model proposed in [74] to infer system throughput. We use a 2-layer architecture in the neural network model. The input layer is connected with two hidden layers with 24×12 neurons. The model uses ReLU as the activation function and a learning rate of 0.01. The model output are the execution times of collocated jobs, which we use to calculate the resulting system throughput. To fit the outputs for execution time prediction, the model removes the `softmax` function at the output layer which is usually used for classification. The model inputs and outputs are shown in Figure 5.3.

Our model uses extra inputs including jobs' CPU and memory power consumption, and the performance ratio between using all and half of the cores $\frac{\texttt{perf}_{\texttt{all}}}{\texttt{perf}_{\texttt{half}}}$. These additional inputs help to increase the model accuracy. Except for the performance ratio, all other inputs are collected when the job runs exclusively on half of the cores distributed across sockets without a power bound. Our model output is execution time of collocated jobs, which we use to calculate the resulting system throughput. The model inputs and outputs are shown in Figure 5.3. Table 5.1 lists the details of

metrics as input for interference prediction.

Table 5.1: Metrics used in CAPS for workload interference prediction. The metrics are platform specific and may be subject to change on other platforms.

Events type	*Description*
memory hierarchy events	*Memory read bandwidth* *Memory write bandwidth* *Local L3 cache miss* *Remote L3 cache miss* *L3 miss rate* *L3 request rate* *L2 miss ratio*
instructions related events	*Cycles per instruction* *Unhalted time / runtime*
power consumption data	*CPU power* *DRAM power*
performance ratio	$perf_{all}/perf_{half}$

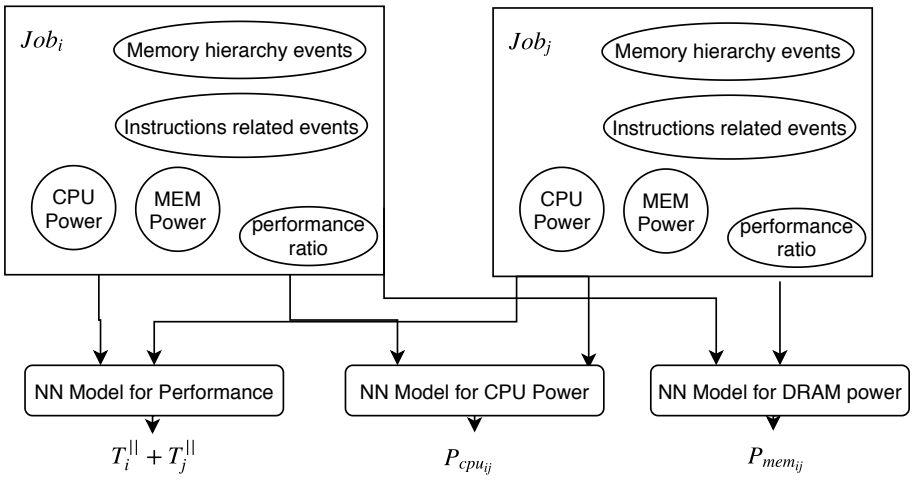


Figure 5.3: The performance and power estimation models.

Based on the model predicted system throughput, we categorize job pairing into three groups in Table 5.2. The scheduler only collocate complementary jobs that achieve $\geq 1.2X$ throughput. We do not collocate interfering jobs as their collocation degrades throughput. If the predicted throughput is $[1.0, 1.2]X$, we run the jobs in sequence by default. This scheduling decision misses optimization opportunities, but is simple and avoids potential degradation due to model misprediction.

Contentions between jobs affect node and components' power consumption. Naively, we might assume the CPU or memory power of collocated jobs is the sum of each job's power subtracted

Table 5.2: Throughput of job pairing and collocation decision under abundant power ($P_b = P_\infty$)

Predicted STP	Job pairing	Collocate?
> 1.2	Complementary	Yes
< 1.0	Interfering	No
$[1.0, 1.2]$	Neutral	No

by the component's base power. Nevertheless, contentions decrease the performance of each job, which in turn reduces the job's power demand on CPU and memory.

We assume the power demands of collocated jobs are related with each job's power profile and their performance loss due to contention. In other words, they are positively correlated to the job's power profile and negatively correlated to the performance loss. Based on this assumption, we build different neural network models, and train them the same performance metrics to infer the resulting CPU and memory power demands ($P_{cpu_{ij}}$, $P_{mem_{ij}}$) for collocated jobs, as shown in Figure 5.3. These models use the same activation function and learning rates as the STP neural networks model. We utilize a $36 \times 24 \times 12$ and 24×12 hidden layer architectures to predict CPU and memory power respectively.

5.2.2 Contention vs. Power Coordination

When P_b is limited, $P_b < P_{cpu_{ij}} + P_{mem_{ij}}$, and the collocated jobs are able to use up all the available power budget. The key question is how to allocate this power budget P_b between components to maximize system throughput.

We find out that for the same total power budget P_b, there exists a certain power distribution between CPU and Memory, $(P_{b,cpu}, P_{b,mem})$, that minimizes contention and maximizes throughput. Here, we take ECP Proxy applications CoMD and CloverLeaf collocation as an example. We reduce the total budget from a certain value, and compare the throughput between two scenarios: (1) budget cut performed on CPU, and (2) budget cut performed on Memory. We observe from Figure 5.4 that throughput decreases gradually as CPU power budget decreases. In contrast, throughput decreases dramatically as memory power budget decreases, suggesting that memory power cut reduces the memory's capacity for data transfer and severely aggravates memory contention.

These results indicate that memory power has greater impacts on system throughout of collocated jobs. Based on these results, we adopt the following power allocation strategy: given a limited power budget, we allocate a sufficient amount of power to memory firstly to avoid exacerbated contention and allocate the remaining power to CPU.

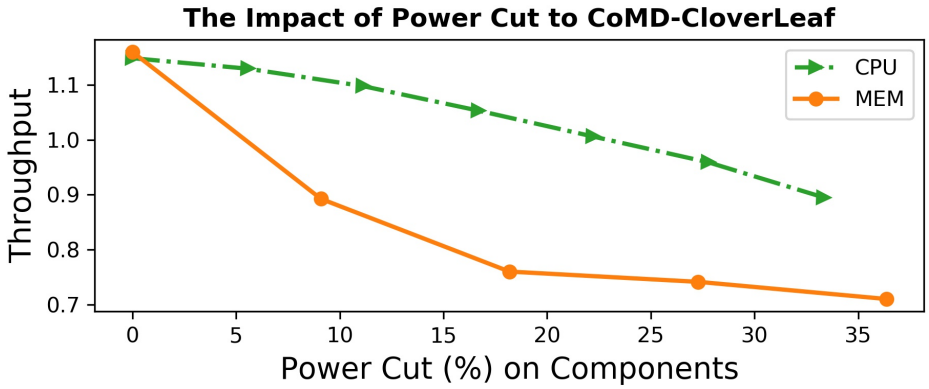


Figure 5.4: Impacts of power coordination between CPU and DRAM on workload throughput.

5.2.3 Contention under Power Limiting

Assume the power budget would be optimally distributed between CPU and memory. In this subsection we investigate how STP varies with power budget P_b.

Figure 5.5 illustrates that STP generally decreases as power budget P_b decreases. As the power budget drops from abundant to inadequate, interfering jobs become more contentious, while complementary jobs may start to contend due to reduced capacity of resources and eventually interfere with each other. This trend indicates that power limiting aggravates contention between collocated jobs.

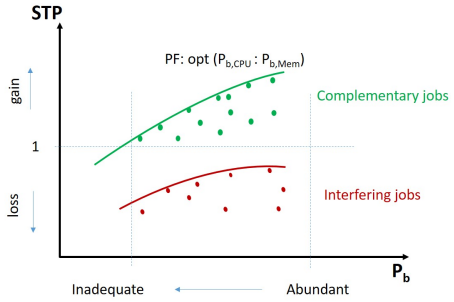


Figure 5.5: STP vs. power budgeting. Throughput decreases as the total power budget P_b decreases. Complementary jobs may become contentious when the power budget is inadequate.

A scheduling question in power bounded computing is: for a given power budget P_b and a pair of jobs i and j, should the jobs be collocated for the sake of throughput? Based on the trends in Figure 5.5, we classify job paring into three cases and propose the following strategies, where $STP(P_\infty)$ is the system throughput under abundant power.

- Case I: $STP(P_\infty) < 1.0$. Never collocate jobs i and j for any given power budget.
- Case II: $1.0 < STP(P_\infty) < 1.2$. Don't collocate jobs i and j for any given power budget. This strategy may lose some opportunities for throughput improvement but is simple.
- Case III: $STP(P_\infty) > 1.2$. Collocate jobs i and j if the given power budget $P_b > P_{th}$, where P_{th} is a threshold we choose heuristically. This threshold ensures collocating the jobs has a system throughput greater than 1.

We now explain in detail how we determine the threshold P_{th} for a pair of jobs i and j in case III. One approach is to measure `STP` under all possible power budgets P_b and uses the collected data to locate the Pareto frontier and its intersection with $\texttt{STP} = 1$ in the $\texttt{STP} - P_b$ space. This intersection point points to P_{th}. This approach requires extensive profiling and is not suitable for online usage.

Instead, we leverage the characteristics of complementary jobs and use a heuristic method to estimate P_{th}. Specifically, a pair of complementary jobs typically consist a compute-intensive, scalable job, and a memory-intensive, non-scalable job. When they are collocated, there exists a range of power budget where power reduction linearly decreases the performance of compute-intensive jobs but only slightly changes that of memory-intensive job. Prior work [25] suggests that we can use a linear function to model CPU power with CPU frequency. Assuming $\frac{T_i^{|}(P_b)}{T_i^{||}(P_b)} \approx \frac{1}{2}$ for the compute intensive job i, and the beneficial overall throughput is 1.2, then $\frac{T_j^{|}(P_b)}{T_j^{||}(P_b)} \approx 0.7$ for the collocated job j, which is the same when $P_b = P_{th}$. Ensuring $\texttt{STP}(P_{th}) \geq 1$ requires $\frac{T_i^{|}(P_{th})}{T_i^{||}(P_{th})} \geq 0.3$. Thus P_{th} can be estimated as the power when CPU frequency is $0.6 \cdot f_{\texttt{CPU}_{\texttt{max}}}$.

5.3 CAPS Design and Implementation

To mitigate the contention between collocated workloads under a power bound, we develop a contention aware scheduling with a two-level power coordination that optimally allocates power between nodes and components. The scheduling framework is shown in Figure 5.6. Once a job finishes, the scheduler reclaims the released hardware resource and power budget. It further selects to collocate another complementary job from the queue if such a job exists.

In Chapter 3, we have found that the CPU power allocation should fall within a range denoted by $[P_{\texttt{cpu}_L}, P_{\texttt{cpu}_H}]$ to achieve reasonable performance and power efficiency. Here $P_{\texttt{cpu}_L}$ and

$P_{\texttt{cpu}_H}$ are the power consumption for the job when CPU runs at the lowest and highest frequencies respectively. We apply the same principle for job collocation. Specifically, we use the neural network models to estimate $P_{\texttt{cpu}_{H_{ij}}}$ and $P_{\texttt{cpu}_{L_{ij}}}$ from jobs' performance and power profiles when CPUs run at the highest and lowest frequencies. We further use these two values to estimate the threshold power $P_{\texttt{th}_{ij}}$ suitable for their collocation to avoid throughput loss. We always allocate sufficient power to memory to avoid significant performance degradation, and denote this power as $P_{\texttt{mem}_{ij}}$.

Assume job j is assigned to run on m nodes and only uses a subset of the cores, and the available power for these nodes is P_r. The scheduler examines the next job in the queue. To efficiently utilize the power budget, the scheduler have different job scheduling and power allocations based on the predicted power consumption of the collocated jobs.

- $P_\texttt{r}/m >> P_{\texttt{cpu}_{H_{ij}}} + P_{\texttt{mem}_{ij}}$: the scheduler may switch another job to co-run with j to avoid power budget waste or request system to reclaim extra power.
- $P_\texttt{r}/m > P_{\texttt{cpu}_{H_{ij}}} + P_{\texttt{mem}_{ij}}$: the scheduler shifts extra power to other nodes as illustrated in following subsection.
- $P_\texttt{r}/m > P_{\texttt{th}_{ij}} + P_{\texttt{mem}_{ij}}$: the scheduler allocates $P_{\texttt{mem}_{ij}}$ to memory, and the remaining power to processors, where $P_\texttt{th}$ is the model estimated power threshold suitable for job collocation.
- $P_\texttt{r}/m < P_{\texttt{th}_{ij}} + P_{\texttt{mem}_{ij}}$: the scheduler may choose to run job j by itself to avoid significantly performance loss by insufficient power.

After managing job scheduling and power allocation within nodes, the scheduler may coordinate power among multiple nodes to achieve global optimal throughput, and ensure all jobs allocated with their acceptable power ranges.

Figure 5.6 presents the two-level power coordination framework. The scheduler obtains the power demands of each job pairing from the power prediction models and coordinates power within nodes. It groups nodes to run the same jobs and shifts power between groups to meet different jobs' power demands. The scheduler could also shift power inside a group if manufacture variability exists as in [15].

Once a collocated job completes execution, *CAPS* coordinates power at the node and cluster levels to best fit workloads' power demand. It does not consider the phase changes inside a job to control overhead.

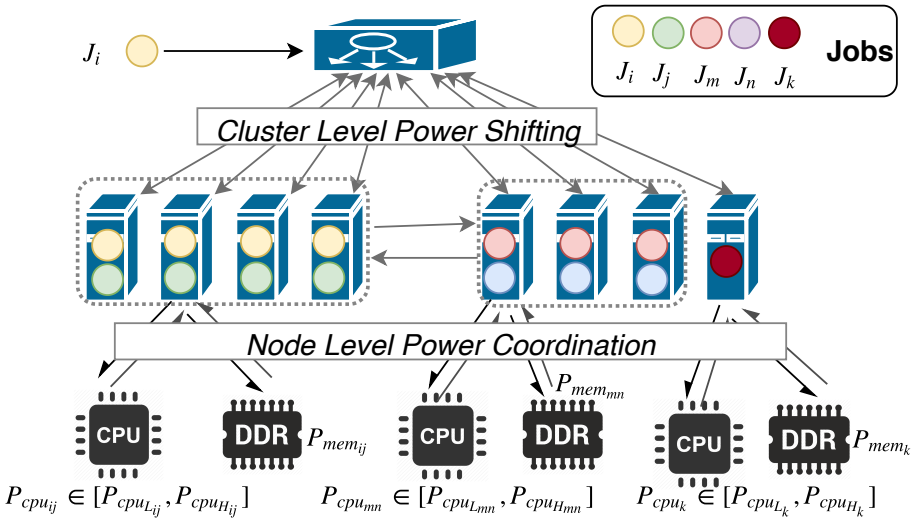


Figure 5.6: The two level power coordination framework for job collocation.

5.4 Evaluation

5.4.1 Experimental Setup

The evaluation is implemented on the 8 dual Haswell Processors nodes. The benchmarks used in the evaluation are listed in Table 5.3. Each application are running with 24 MPI processes or 24 threads/1 MPI process on a node solely in the scheduling approach FCFS, Cord, POW. In Twins and CAPS, we run each workload with 12 MPI processes or 12 threads/1 MPI process concurrently with another workload on a node.

5.4.2 Performance Models Validation and Accuracy

We train the models for interference and power predication described in Sections 5.2 using training data set collected from the benchmarks and assess the model accuracies using the test data set. Overall, we obtained the results:

- After having examined multiple neural network models, we find that the model of two hidden layers, with 24 and 12 neurons at the two layers, provides the best accuracy. Thus, we adopt such network in CAPS for interference prediction.
- The average estimation error of the interference model is about 7%. The model tends to underestimate the interference for some memory intensive workloads.

Table 5.3: The list of benchmarks used to evaluate CAPS.

Benchmark	Problem size/Input	Scalability	Intensity bias	Source	Power*
BT-MZ	D	low	compute	NPB	170
SP-MZ	D	low	memory	NPB	177
EP	D	high	compute	NPB	158
AMG	-n 480 480 240 -laplace -solver 2	high	memory	CORAL2	180
LULESH	size=70	medium	compute	CORAL2	173
STREAM†	Array size = 2e9	low	memory	STREAM	230
CloverLeaf†	cells = 15360×16360	medium	memory	ECP proxy	230
TeaLeaf†	cells = 6000×6000	medium	memory	ECP proxy	228
CloverLeaf3D†	cells = $384 \times 384 \times 384$	medium	memory	ECP proxy	190
TeaLeaf3D†	cells = $512 \times 512 \times 512$	medium	memory	ECP proxy	220
MiniFE	-nx 512 -ny 512 -nz 512	medium	compute	ECP proxy	165
MiniMD	-s 90	high	compute	ECP proxy	150
MiniAero	-s 20	high	compute	ECP proxy	154
CoMD	Atoms = 3.2e7	high	compute	ECP proxy	150

*: Power consumption is obtained while run exclusively (solo) on all cores (watts)
†: They are extreme memory intensive workloads. They significantly interfere each other if co-scheduled on the same node.

- The average difference between the prediction power and actual power is less than 10 watts. Thus, these power models satisfy the need to estimate the co-scheduling jobs power consumption with a high accuracy.

5.4.3 Workload Sharing Recommendations

We also find the following rule of thumb with regard to node sharing from our model prediction and experimental results.

- Extreme memory intensive applications like `STREAM`, `CloverLeaf`, `TeaLeaf`, `CloverLeaf3D`, `TeaLeaf3D` significantly interfere each other. It is not recommended to share a node between these applications.

- Less memory intensive application cause little interference to extreme memory intensive applications. Conversely, extreme memory intensive application interfere others significantly. Nevertheless, the overall throughput increases. Thus co-scheduling extreme memory intensive application with others applications increases system throughput.

- Compute intensive applications and weak memory intensive applications cause low interference to each other. However, the memory bandwidth could be underutilized when two compute intensive applications are co-scheduled.

5.4.4 The Performance of the CAPS scheduler

To evaluate the performance of the CAPS scheduler, we compare it with other four methods both without power limit and under multiple power bounds. The key differences between evaluated methods are summarized in Table 5.4.

Table 5.4: Comparison of used strategies to improve throughput under power bounds

Name	Power Coordination		Co-Scheduling	Contention Aware
	cross-components	cross-nodes		
FCFS	✗	✗	✗	✗
Cord [42]	✓	✗	✗	✗
POW* [34]	✓	✓	✗	✗
Twins†	✓	✓	✓	✗
CAPS	✓	✓	✓	✓

*: POW [34] originally does not support cross-components power coordination. We pair it with Cord [42] to compare fairly under power bounds.
†: We also integrate Cord [42] and POW [34] to Twins to demonstrate the importance of contention-aware co-scheduling.

- **FCFS**: First-come, first-serve (FCFS) is one of the most commonly used scheduling policies on production HPC systems. As FCFS doesn't manage power, we adopt a fixed power allocation in the experiments. Specifically, we allocate no less than 40 watts of power to memory, which is sufficient for most applications in our experiments, and allocate the remaining power budget to CPUs. FCFS doesn't co-schedule jobs either and thus we use all the cores on a node to execute the same job for performance.

- **Cord** [42] as described in Chapter 3. Cord coordinates power allocation between CPUs and memory based on the workload characteristics. As it does not support node-sharing, we use all cores on a node for a job.

- **POW** [34]. POW dynamically shifts power between nodes according to the power demands of running applications on each node. As POW doesn't support power coordination between components at the node level, we pair POW with Cord to support inter-node and intra-node power shifting.

- **Twins**. To demonstrate the importance of contention aware co-scheduling, we investigate the performance of co-scheduling a replica of the same workload on a node similar as [27]. At the same time, we integrate cross components and cross nodes power coordination to Twins.

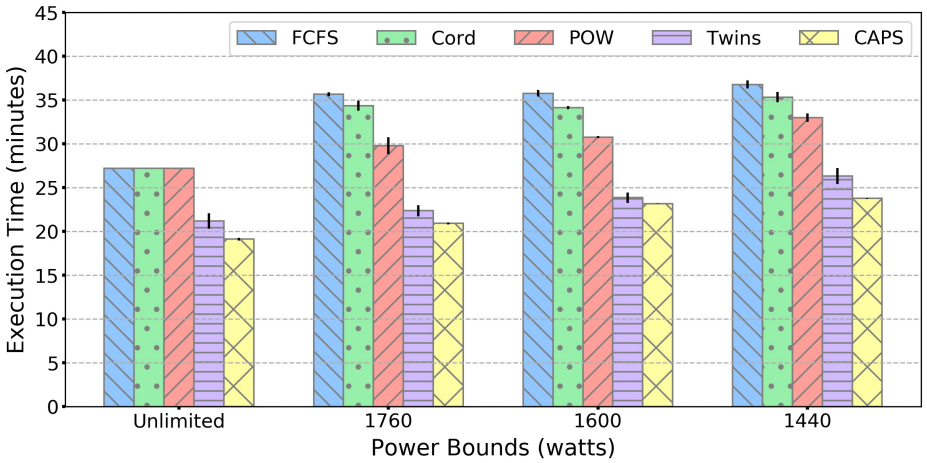


Figure 5.7: System throughput comparison. The figure shows the execution time for all the jobs under different power budgets. Less time indicates better performance.

- **CAPS**. Our proposed scheduler allows jobs to co-run on nodes to maximize the utilization of hardware resource and power budget. It may adjust job's node level concurrency to accommodate job co-running based on application scalability, map affinity, performance interference and available power budget. Simultaneously, it coordinates power allocation within nodes and across nodes.

5.4.4.1 System Throughput

Figure 5.7 shows the execution time under the five methods. CAPS reduces the execution time by 25% when the power is unbounded. Without power bounds, Cord and POW perform about the same as FCFS. In contrast, Twins increases the system throughput by 20%. We observe that Twins increases the throughput significantly for less memory intensive applications. For example, running `miniFE` with 24 threads solely on 4 nodes takes 135 seconds and running two `miniFE` with each one owns 12 threads on 4 nodes takes 150 seconds. For workloads that achieve low scalability on all cores but acceptable scalability on half cores, Twins also increases the system throughput. For extreme memory intensive workloads, running a replica doesn't benefit the system throughput. In almost all cases, CAPS performs better than Twins because it avoids co-scheduling workloads with high interference to the same node. Overall, CAPS increases the system throughput 10% against Twins.

When power bounds are enforced, CAPS increases system throughput by 30%. Under the

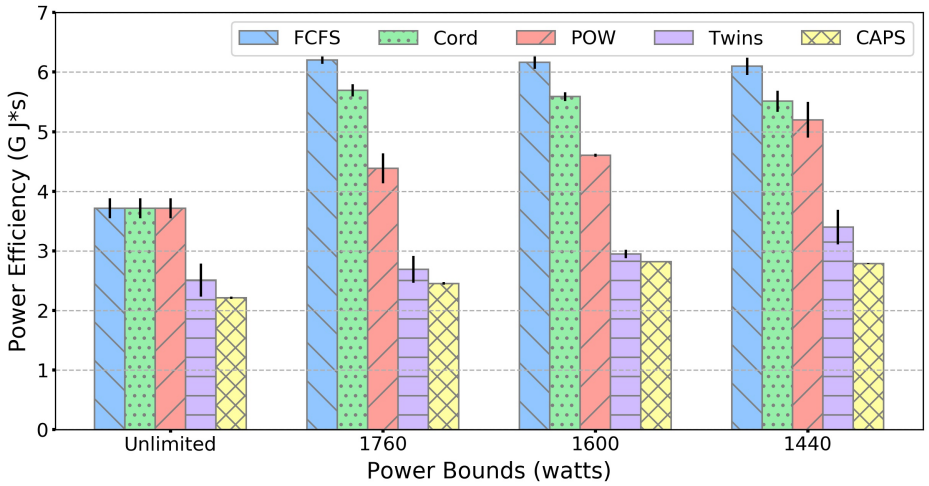


Figure 5.8: Power efficiency comparison. This figure shows the Energy-Delay products. Smaller values mean higher efficiency.

1600 watts power bound, FCFS allocates 40 watts to memory and 160 watts to CPU on each node. Cord coordinates the power between CPU and memory according to applications power demands and increases the throughput about 5%. Even though, the power cap impacts workloads significantly when these workloads consume more than 200 watts on each node (see Table 5.3). POW shifts power between workloads and ensures that workloads on different nodes get sufficient power. Comparing with unbounded power, POW only increases elapsed time about 10% and saves more time compared with FCFS under the same power bound. POW's results demonstrate appropriate power coordination cross nodes and cross components can significantly increase system throughput. Because CAPS considers workload contentions and schedules jobs to achieve higher power utilization, it performs consistently better than other methods on power-bounded system.

5.4.4.2 Power Efficiency

Energy-delay product provides a easy-to-measure metric for power efficiency. The results summarized in Figure 5.8 demonstrate that in comparing with FCFS, CAPS increases power efficiency by more than 40% without power bound and more than 50% with multiple power bounds. Meanwhile, CAPS saves more than 25% energy consumptions to complete all jobs. The reason that CAPS provides much better energy efficiency is that CAPS actively manages power allocation, reducing wasted power budget or overhead and shifts more power towards jobs that are power

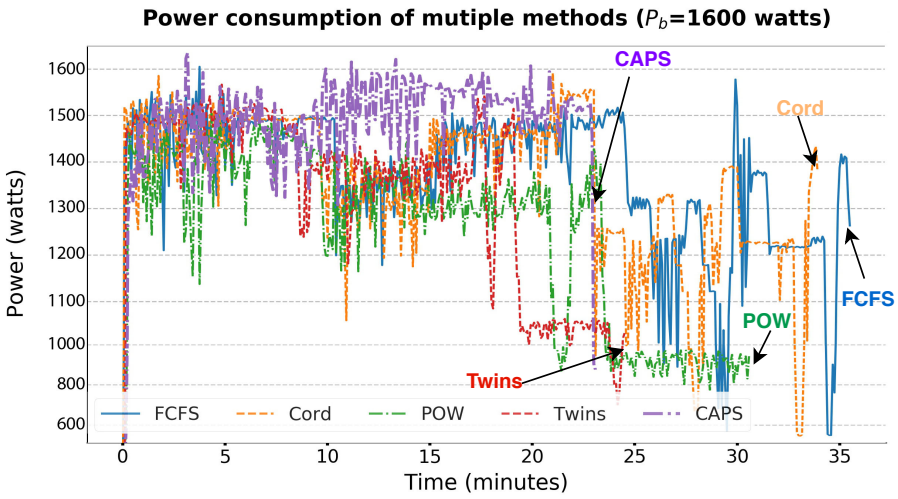


Figure 5.9: System power traces. The figure shows that CAPS maintains a high power utilization at a constant level.

greedy. These results also demonstrate power-bounded scheduling improves both power utilization and energy efficiency.

5.4.4.3 Power Utilization and Stability

Figure 5.9 provides the system power traces for the five scheduling methods. Given a power bound at 1,600 Watts, CAPS maintains the power consumption over 1,500 Watts most time and over 1,400 Watts for the rest. This result proves that CAPS accomplishes its objective of increasing the power budget utilization and also keeps the power at a more stable level.

Further, we plot the power traces of all eight nodes in Figure 5.10. We observe that as CAPS co-schedules workloads together with power allocation at the node level, the power consumptions on all the nodes are close to the average node power budget. In short, CAPS leads to more balanced power distribution across the nodes than POW and Twins, the other two power bounded methods.

5.4.4.4 Other Metrics

Table 5.5, 5.6, 5.7, 5.8 summarized some other metrics of different methodology. As seen from the tables, all methods' average power consumption is below the given power budget. CAPS outperforms other methods both on energy and energy delay significantly.

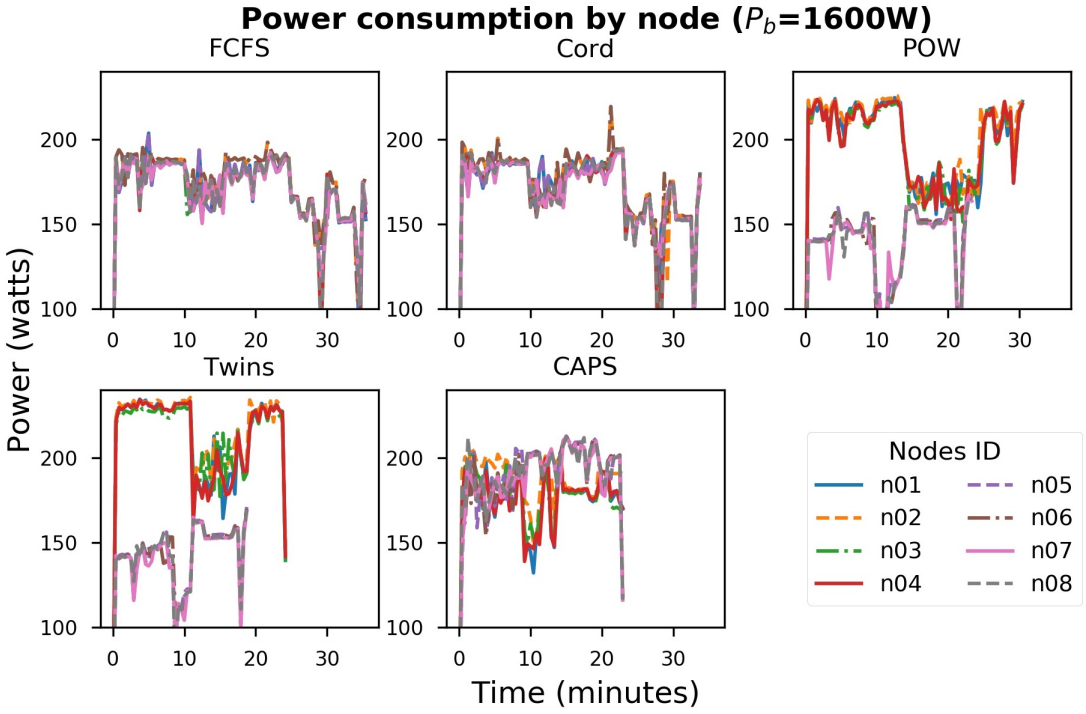


Figure 5.10: Nodal power traces. CAPS reduces the imbalance of power distribution between nodes comparing with the other two power bounded methods (POW, and Twins).

Method	Exec. Time(s)	Ave. Power (watts)	Max Power (watts)	Energy (Joules)	Energy×Delay
FCFS	1633	1358	1717	2.3e7	3.7e10
Cord	1633	1358	1717	2.3e7	3.7e10
POW	1633	1358	1717	2.3e7	3.7e10
Twins	1272	1592	1886	2.0e7	2.5e10
CAPS	1147	1667	1862	1.9e7	2.2e10

Table 5.5: Metrics of compared methods without a power bound.

Method	Exec. Time(s)	Ave. Power (watts)	Max Power (watts)	Energy (Joules)	Energy×Delay
FCFS	2140	1354	1724	2.9e7	6.2e10
Cord	2061	1340	1746	2.8e7	5.6e10
POW	1787	1373	1592	2.5e7	4.4e10
Twins	1342	1491	1702	2.0e7	2.7e10
CAPS	1255	1557	1743	2.0e7	2.5e10

Table 5.6: Metrics of compared methods under power bound of 1760 watts.

5.5 Summary

In this chaper, we present CAPS, an application of the power-bounded computing approach to modern clusters which supports job collocation. We build performance models to predict workload interference due to resource contention and power limiting. The contributions of this work are highlighted as follows:

First, we provide methods and techniques, which comprise performance and power modeling, power coordination, and workload-awareness, to improve performance on multicore systems with limited power budgets.

Second, we study and model the relationship between resource contention, power limiting, and performance interferences for collocated workloads.

Third, we design a heuristic contention-aware power bounded scheduling (CAPS), which uses a two-level scheme to distribute power among nodes and components to mitigate contentions between collocated jobs.

Method	Exec. Time(s)	Ave. Power (watts)	Max Power (watts)	Energy (Joules)	Energy×Delay
FCFS	2147	1338	1756	2.9e7	6.2e10
Cord	2048	1332	1682	2.7e7	5.6e10
POW	1847	1350	1551	2.5e7	4.6e10
Twins	1432	1438	1626	2.1e7	2.9e10
CAPS	1391	1456	1648	2.0e7	2.8e10

Table 5.7: Metrics of compared methods under power bound of 1600 watts.

Method	Exec. Time(s)	Ave. Power (watts)	Max Power (watts)	Energy (Joules)	Energy×Delay
FCFS	2206	1252	1696	2.8e7	6.1e10
Cord	2119	1227	1593	2.6e7	5.5e10
POW	1979	1325	1462	2.6e7	5.2e10
Twins	1580	1358	1504	2.1e7	3.4e10
CAPS	1426	1368	1564	2.0e7	2.8e10

Table 5.8: Metrics of compared methods under power bound of 1440 watts.

Finally, we evaluate *CAPS* on an experimental cluster using a set of scientific applications. Results demonstrate that *CAPS* improves system throughput by 10% or more, depending on the power limits, than job collocations that are oblivious to power limits, and 25% or more than the typical first-come-first-serve scheduling deployed on clusters.

CAPS represents a trend of using hardware and software co-optimization and workload-aware system reconfiguration in HPC system design and production. CAPS paves the ways for future HPC system to schedule hardware resource, power, and jobs together for higher system throughput. Future job scheduling system shall co-optimize with power scheduler in a fine-grained way to maximize system throughput under a power budget.

Chapter 6

Online Dynamic Performance & Power Management

The race for power efficient computing drives the integrating of accelerators (such as GPUs and FPGAs) in data center and supercomputers. As of November 2019, 5 of the top 10 fastest supercomputers are equipped with GPU accelerators [3]. The first exascale supercomputer – Aurora deployed in Argonne National Laboratory will also integrate GPU cards. These GPU-accelerated systems not only support traditional HPC applications, but also support emerging deep learning in various disciplines including science, imaging processing, and natural language processing. As GPU cards are the major power consumers in such systems [5], improving energy efficiency for GPU applications is an urgent issue.

As introduced before, dynamic voltage and frequency scaling (DVFS) is effective for managing power and meeting performance goals. DVFS technology is available on both GPU cores and memory. GPU can transit among multiple performance states, each associated with a pair of voltage and frequency via DVFS. Scaling down frequency can decrease performance linearly for compute-intensive applications but less or little for others with insufficient parallelism or/and intensive memory accesses. However, scaling down frequency always decreases power more than linearly for any application.

The challenge of optimally using DVFS lies in its requirement for prior knowledge of applications and their performance and power profiles. For a given application and user specified perfor-

mance goal, minimizing power requires the knowledge of the mapping of frequency $f \rightarrow (\texttt{perf}, \texttt{power})$ for all possible DVFS settings, and identifying the Pareto frontier in the performance-power space. Such knowledge can be obtained with extensive application profiling. However, due to a large number of combinations of GPU core and memory frequencies, extensive profiling is time consuming, not suitable for online employment for unknown workloads. To quickly identify the optimal settings, prior works [39, 105, 66, 69, 75] instead analyze source codes or uses hardware performance counters to obtain fine-grained workload profiles. However, such fine-grained profiling and source code transformation are impractical to employ and not transparent to users.

In this chapter, we focus on iterative workloads, which are common in both HPC systems and commercial data centers. Iterative workloads repeat (roughly) the same computations, typically for a large number of times. For example, training a Convolution Neural Networks (CNN) model can involve millions iterations to obtain model parameters. Each iteration includes forward propagation from the first layer, to middle layers until the output layer, and back propagation to the first layer. Similarly, many HPC scientific simulations iterate the same computations until results converge. The repeated iterations generate (roughly) periodical patterns of power, GPU core utilization, GPU memory utilization, as demonstrated traces of *Inception3* [98] CNN model training in Figure 6.1.

We leverage the periodic patterns in resource utilization and take a novel approach to automatically extract performance and power indicators. Here *indicators are the mapping of* $f \rightarrow (\textit{perf}, \textit{power})$ for a few of the available GPU frequencies. Our rationale is: when frequency changes, the observed period change indicates the resulting performance impact for the running application. Based on this rationale, we strategically profile the traces for several iterations, and then uses Fourier transform on these traces to obtain the mapping for a few frequencies. Such profiling is lightweight and practical, and executing Fourier transform on the short traces incurs minimal cost. To ensure robustness, we extract the indicators from multiple resource utilization traces including power, GPU core and memory utilization.

A few indicators are insufficient to identify the Pareto frontier in the performance-power space for the workload under study. To address this issue, we take a novel hybrid modeling approach to robustly and accurately predict performance and power from small profiling sample sizes. Specifically, we build a piecewise linear analytical model between speedup/power and GPU frequency using the obtained indicators. This model is simple but inaccurate. To improve accuracy and robustness, we build non-linear neural network models that use the analytical model as one feature, and also

include other features learned form similar workloads. We investigate two neural networks: fully connected multilayer perceptron (MLP) and recurrent neural network (RNN). MLP has been suggested for accurate performance modeling and prediction [94], while RNN is capable of capturing features from resource utilization traces. To minimize online training time, we pre-train the models offline with a set of benchmarks and transfer them to learn with the analytical model. The resulting hybrid model is then used to predict performance and power under all configurations and identify the Pareto frontier.

In this chapter, we present the design and implementation of an online dynamic performance-power (ODPP) management framework. ODPP runs on the host and quickly adjusts GPU DVFS configurations to meet user specified performance and power objectives for GPU workloads. It is automatic, non-intrusive, and transparent, and supports unseen applications. ODPP uses two novel ideas. First, it extracts performance and power indicators from easily obtainable resource utilization traces of iterative workloads. Second, it builds hybrid models to quickly learn the workloads from small training samples, particularly suitable for online deployment. We implement a prototype and evaluate it on a GPU-accelerated system. Results show that ODPP improves energy efficiency by over 30% than the nominal (highest GPU frequency) configuration. It also improves performance by 8% on average under multiple power bounds.

6.1 Characteristics of Iterative workloads

Dynamically identifying the optimal frequency configuration balancing performance and energy is quite challenging for HPC runtime management, due to (i) the lack of appropriate performance indicators which should be *accurate*, *lightweighted*, and *easy to measure*; ideally *non-intrusive* and *transparent* to the workloads; (ii) infeasible to traverse the huge configuration space for global optimal at runtime without offline pre-profiling or prior knowledge.

For iterative workloads, the underlying periodic execution pattern can bring three very useful features for the aforementioned two problems: (a) If we can precisely extract the major iterative period T, we can directly predict the performance speedups, since an iterative application's overall execution time is almost in proportional to T. Therefore, by tracking the variation of T, we can verify whether a particular configuration can satisfy the users' QoS demand. (b) The periodic pattern can also be reflected in some profiling signals that can be sampled online without much overhead, such as

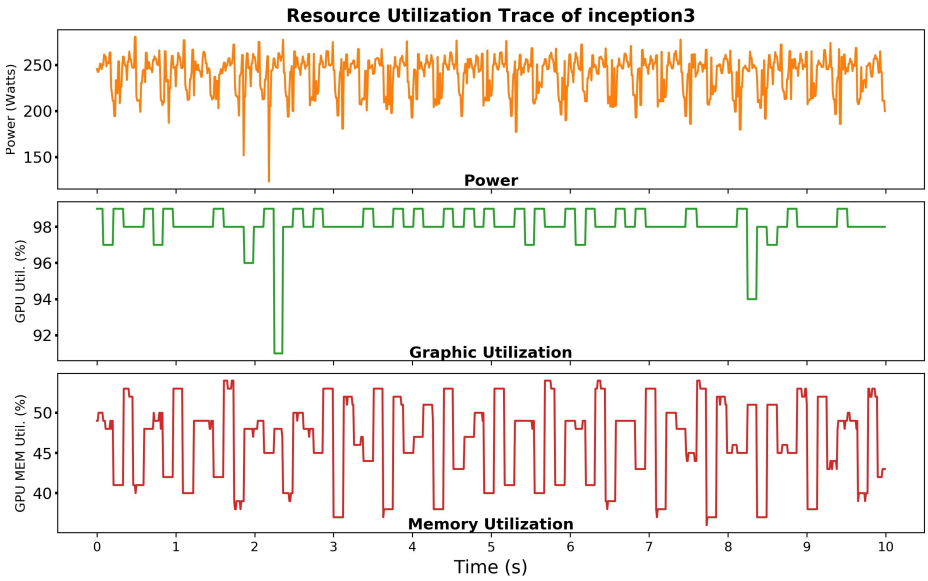


Figure 6.1: A 10s screen-shot of the GPU power, core utilization, memory utilization trace for training *Inception3*. The trace is obtained on NVIDIA Titan-XP while running Inception3 CNN model with ImageNet dataset and 64 images per batch.

core & memory utilization profiles and power trace. These signals are collected in the system level, which is isolated from the running workloads. (c) Since the application is iterative, we can establish an autotuning practice: once we verify that the application is iterative and has entered the steady state of iterative execution, we can use the first few periods for sampling and model-calibration, and predict the optimal configuration at the first trial. We then test this configuration and adjust the model based on performance/power feedbacks. After several rounds, the model will converge to the optimal configuration, which is used for the remaining periods. In this way, the approach is fully dynamic, being capable of addressing application and system change effectively and automatically. We discuss two examples representing typical machine learning (ML) and HPC applications:

CNN Training: modern deep convolution neural networks (CNNs) are trained in batches using stochastic gradient descent based optimization methods. The processing of each batch, including forward and backward propagation, comprises a full iteration. Figure 6.1 shows the traces of GPU power, core utilization, and memory utilization under 24 iterations (duration≈10s) for training an *Inception3* CNN network [98]. As can be seen, despite with noise and perturbation, all of the signals show clear periodic characteristics. We claim this period matches the processing latency of a batch, because: (a) the processing of all batches are more or less homogeneous, showing repeated manners; (b) the processing within a period is application-specific and mostly non-periodic. In fact, *Inception3*

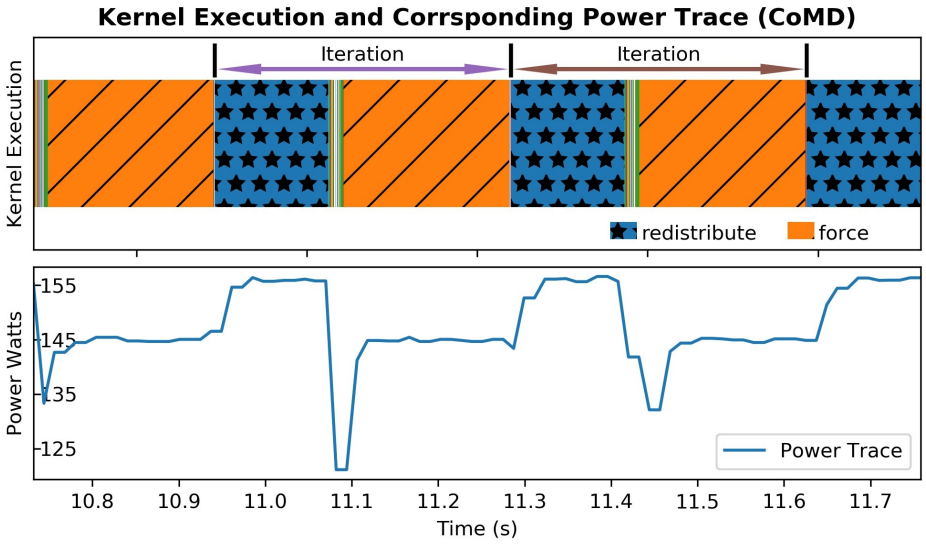


Figure 6.2: *CoMD*'s phase change and corresponding power during two iterations. The trace is obtained on NVIDIA Titan-XP while running CoMD.

includes 13 layer groups, each with distinct computation intensity, reflected in the core utilization trace. The last layer group contains fully connected layers, consuming the peak power shown in the power trace. Finally, there is an burst of DRAM access when fetching image data into GPU's device memory per batch, reflected in the memory utilization trace. Since all of them follow the same period, we can integrate them to alleviate the distortion from noise.

CoMD: *CoMD* is an ECP proxy app representing the common computation and communication patterns of classic molecular dynamics applications. For this HPC application, each iteration includes four phases: *position*, *velocity*, *redistribute*, and *force*. As *position* and *velocity* finished very quickly, the power monitor cannot precisely capture the two phases. However, for *redistribute* and *force*, the phase transition with 10 watts power difference is clearly displayed in the power trace, as shown in Figure 6.2. The periodic pattern depicted in the power trace matches the iteration phase-change very well.

The two examples indicate that an iterative application's periodic execution feature is reflected in the power and resource utilization traces, and can thus be extracted effectively. In the next section, we present the design and implementation of ODPP.

6.2 ODPP Design and Implementation

The ODPP framework achieves performance/power autotuning management through six steps: (i) verifying whether execution enters a stable iterative stage, discussed in Subsection-A; (ii) precisely sampling the traces of performance indicators (i.e., GPU power, core/memory utilization), discussed in Subsection-B; (iii) extracting the period information from the traces, discussed in Subsection-C; (iv) constructing performance/power models for effective DVFS configuration space exploration, discussed in Subsection-D; (v) continuous model calibration and adjustment for optimal configuration through autotuning, discussed in Subsection-E.

6.2.1 Iterative Stage Detection

The execution time of an iterative application can be expressed as:

$$T_{total} = T_{init} + M * T_{iter} + T_{final} \tag{6.1}$$

where T_{iter} is the major steady stage, as M can be very large. T_{init} and T_{final} are transit stages. T_{init} usually configures the execution context and fetches data from storage for preparing the execution whereas T_{final} typically releases the memory and dumps the results. Consequently, the utilization of the computation and memory units, as well as power consumption in T_{init} and T_{final} tend to be lower than in T_{iter}. In addition, T_{init} and T_{final} usually do not show periodic execution patterns. We show an example about *CoMD* in Figure 6.3.

ODPP therefore verifies the iterative stage based on two criteria: (i) surges are observed for all or most resource utilization traces; (ii) periodic pattern is then observed for all or most resource utilization traces.

6.2.2 Sampling Methodology

We show how to more precautiously sample the traces of power, core utilization, and memory utilization using NVIDIA's profiling and device management tool `nvidia-smi`. The problem here is that although a sampling interval can be specified in `nvidia-smi`, we observe that the actual sampling interval can be different. After some investigation, we find that the deviation is correlated to three factors: *the current GPU frequency*, *the specified sampling interval*, and *the utilization of*

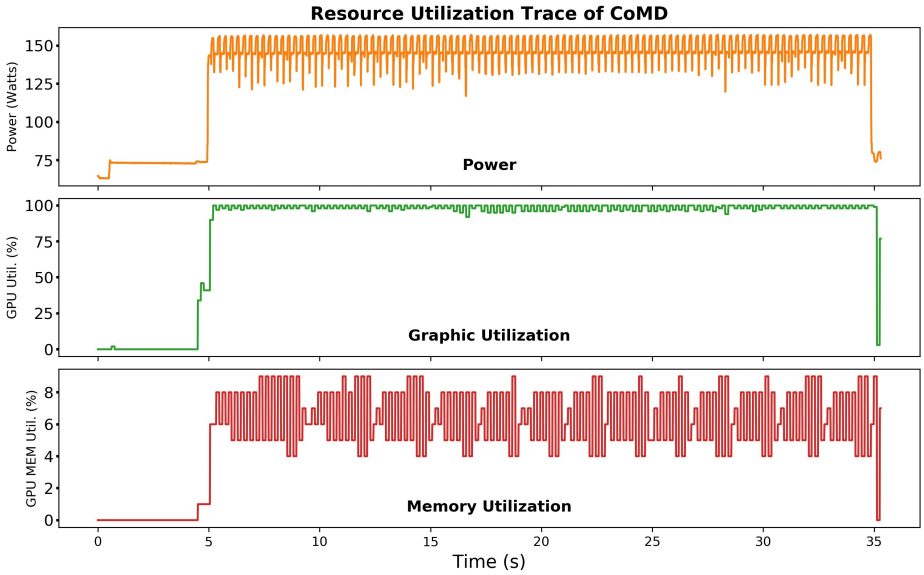


Figure 6.3: The fully trace of running *CoMD*.

the GPU. We define the loss rate of the sampling points as:

$$\texttt{Loss\%} = \frac{d/I_s - N_r}{d/I_s} \tag{6.2}$$

where d is the overall sampling duration, I_s is the specified sampling interval, and N_r is the number of real sampling points actually recorded by the tool. Note, `nvidia-smi` is expected to record $N_e = d/I_s$ sampling points.

Figure 6.4 shows the *loss rate* under different GPU frequencies. As can be seen, if GPU is

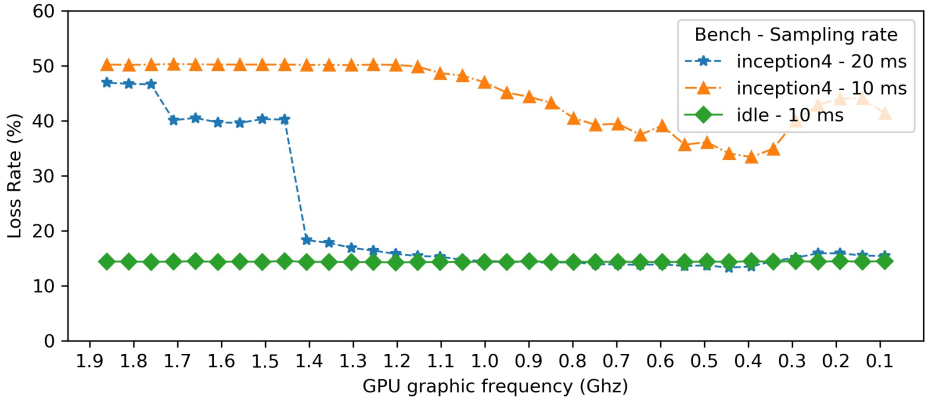


Figure 6.4: Loss rate of `nvidia-smi` sampling under different benchmarks and user specified sampling interval.

idle, the loss rate (green curve) is around 13% when 10ms sampling rate is assigned. However, when GPU is busy, the loss rate (blue curve) increases from right to left under decreasing GPU frequency, to about 45% at 0.1 GHz eventually. In addition, with a lower sampling interval assigned (from 20ms to 10ms), the loss rate (orange curve) also significantly surges. All these deviations do not show obvious patterns, thus is difficult to model directly. Finally, we do not observe any correlation between loss rate and GPU memory frequency.

To correct such a deviation, when having N_r, we use the following equation to revise the original period T_{iter} to eliminate this deviation after extracting T_{iter} in Subsection-C:

$$T_{iter\_corr} = T_{iter} * N_e \ / \ N_r \tag{6.3}$$

6.2.3 Period Extraction and Sensitivity under DVFS

We then focus on T_{iter}. Let us assume the periodic resource utilization can be expressed as a function $\mathcal{F}(t)$. Since $\mathcal{F}(t)$ is a periodic function, it can be expressed as a linear combination of trigonometric series (known as *Fourier Series*) in exponential form as:

$$\mathcal{F}(t) = \sum_{n=-\infty}^{\infty} C_n e^{2\pi jnt/T_{iter.}} \tag{6.4}$$

where C_n are *Fourier coefficients*. The major period T_{iter} can be efficiently extracted from the *frequency domain* of $\mathcal{F}(t)$ via *Fourier transform*:

$$\mathcal{F}(\omega) = 2\pi \sum_{n=-\infty}^{\infty} C_n \delta(\omega - 2\pi n/T_{iter.}) \tag{6.5}$$

the frequency $\mathcal{F}(\omega)$ is a series of impulse functions (i.e., δ) located at $2\pi n/T_{iter}$, with amplitude proportional to C_n. These shifted impulses represent different frequency components in $\mathcal{F}(t)$. The one with the largest amplitude is the major frequency component. By measuring its location shift, we can calculate the period T_{iter}. An example for *Inception3* training is shown in Figure 6.5. Clearly, the major impulse is located around 4 Hz, implying that $T_{iter} \approx 0.25$ seconds.

We then discuss the sensitivity of the period under different DVFS configurations. Figure 6.6 shows the phase-shifting of the major frequency component that corresponds to the period T_{iter} in the combined frequency domain (recall Figure 6.5) by adjusting GPU core frequency under 1.1 GHz,

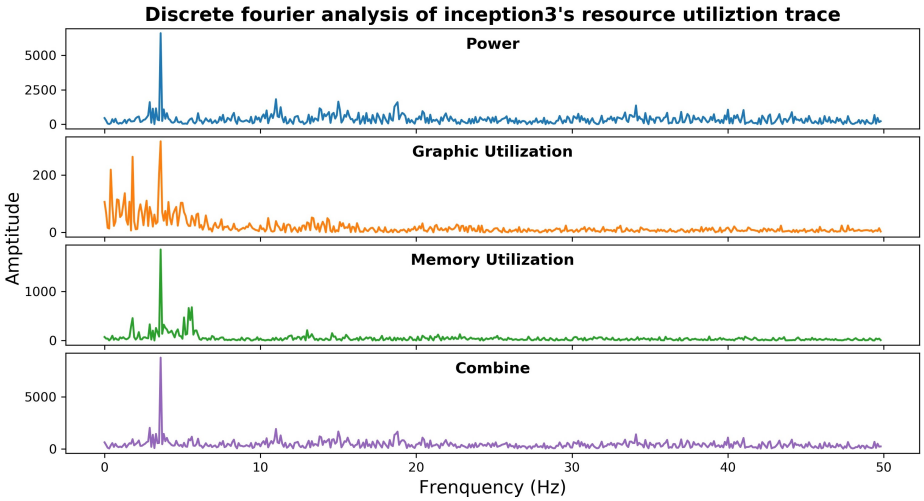


Figure 6.5: The amplitude after transferring the 10s resource utilization trace of Inception3.

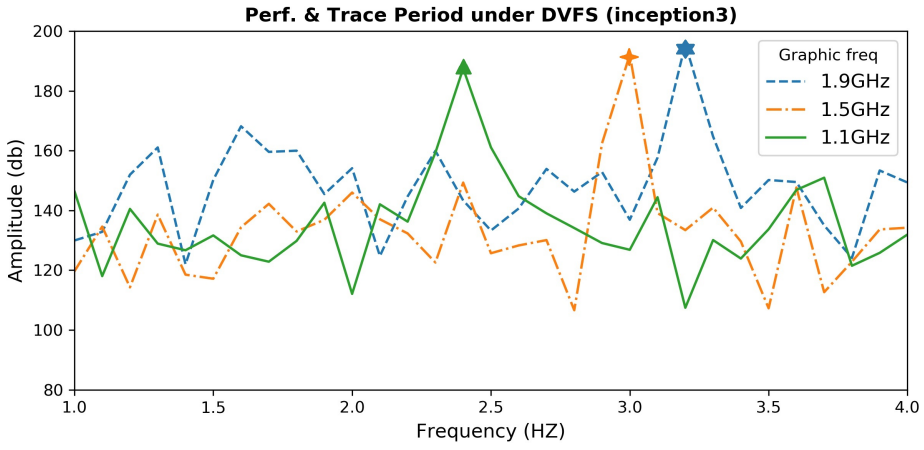


Figure 6.6: The resource utilization trace's FFT Max frequency shifts left while decreasing GPU Graphic frequency.

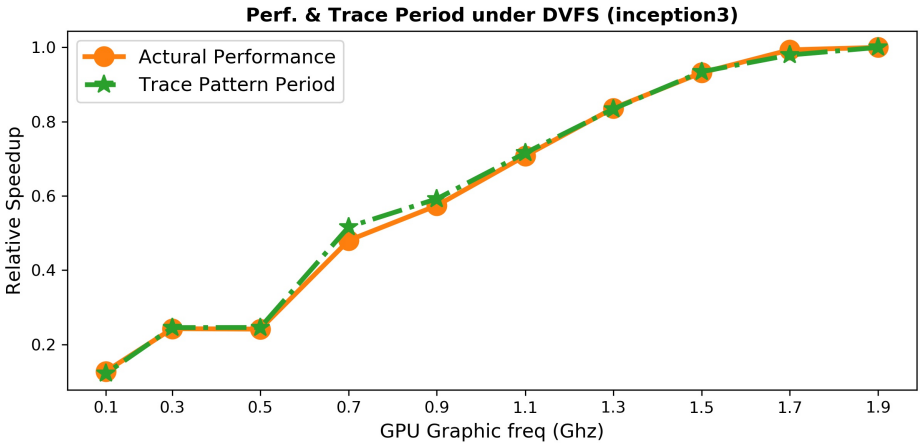


Figure 6.7: The changes of actual performance and period extracted from resource utilization trace with *Inception3*. The base is at the nominal or the highest stable frequency.

1.5 GHz and 1.9 GHz, respectively. This figure shows that the performance impact due to DVFS can be immediately reflected in the frequency domain, and precisely handled by the period extracted.

Now that we can use resource traces and their periods T_{iter_corr} to extract performance indicators, the mapping $f_i \rightarrow (S_i, P_i)$ for one or more specific GPU frequency f_i. Essentially, we can compare performance at frequency f_i and f_0 by comparing periods:

$$S_i = \frac{T_{\texttt{iter\_corr}_0}}{T_{\texttt{iter\_corr}_i}} \tag{6.6}$$

Where f_0 is the base. In our example, we use the nominal GPU core frequency, which is typically the highest stable frequency.

Figure 6.7 compares the speedup extracted from resource traces with measurement for *Inception3* model training under various GPU graphics frequencies. We observe that the extracted period T_{iter_corr} can precisely estimate the application's overall performance. The average deviation between the two curves is less than 1%. Such a high accuracy once again confirms the effectiveness of predicting performance based on periodic information extracted from non-intrusive and transparent power & resource utilization profiles.

Discussions: There are several challenges in accurately extracting T_{iter} from resource utilization trace. The first is the determination of a proper sampling interval when using `nvidia-smi`. A low interval can capture more workload details but at the cost of extra overhead and larger noise (variation in traces). Such noise may skew the amplitude of the frequency components. On the

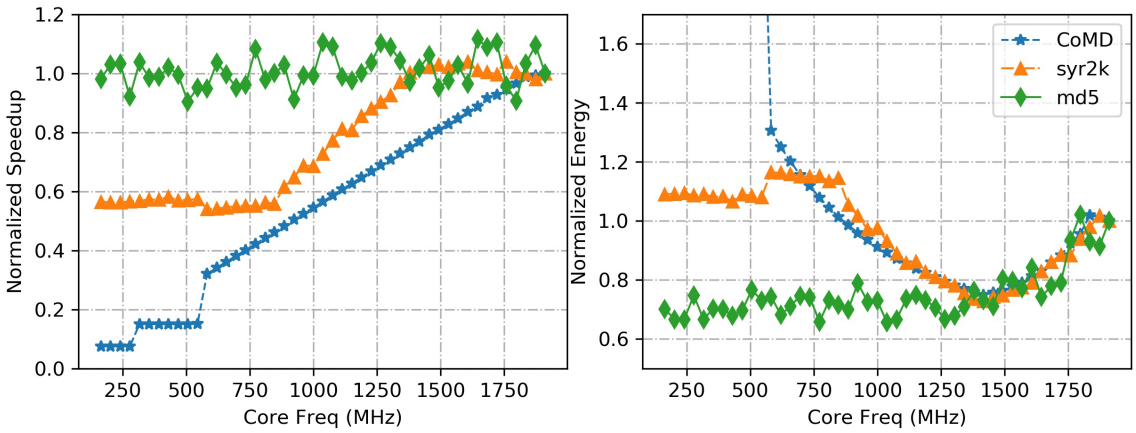


Figure 6.8: Application normalized speedup and energy consumption under different core frequencies.

other hand, a high sampling interval may miss important phase transitions and reduce sensitivity, leading to distorted periodic patterns. To resolve this issue, we try multiple sampling rates at the beginning, and select two that can catch the phase changes without significant noise disturbance. We also cross-validate the extracted period T_{iter} under the two selected sampling rates: a same value indicates the correctness of the extracted period.

The second challenge is the distortion of traces. As shown in Figure 6.5, even though the power and memory utilization traces show a clear major frequency component impulse, the core utilization trace show multiple significant spikes from which a major frequency cannot be identified. To address this issue, we integrate the frequency domain of the three traces to locate one major impulse and calculate the period. Such treatment ensures simplicity of our method. We acknowledge that by dismissing other insignificant pulses we may introduce inaccuracy.

6.2.4 Configuration Space Exploration through Hybrid Modeling

The performance and power impacts of GPU DVFS vary with workload characteristics. Figure 6.8 illustrates the normalized speedup and energy consumption for three applications: *CoMD*, *syr2k*, and *md5*. The performance of *CoMD* scales linearly with GPU frequency in the middle frequency range. However, the performance of *syr2k* keeps stagnant until the core frequency is beyond 850 MHz. It also slightly degrades after the core frequency is above 1.4 GHz, likely due to memory saturation. Alternatively, the performance of *md5* keeps steady. For both *CoMD* and *md5*, the energy consumption dives at higher core frequency until at about 1.4 GHz; after that, it starts

to rise again, forming a sweet point at 1.4 GHz. The energy consumption for *md5* keeps steady until around 1.4 GHz, and then increases.

As can be seen, the variation of performance/power consumption with respect to DVFS is quite complex. Due to the large configuration space (the GPU platform evaluated in Section 6.3 claims supporting 461 different DVFS configurations), it is not feasible to traverse all of them for the global optimal, particularly at runtime for a unknown running workload.

We build a hybrid analytical and neural network (NN) model to predict performance and power in the configuration space. The NN model learns performance and power features from a set of benchmarks and known workloads offline, and retrains its model parameters online according to an analytical model built from the extracted indicators of the running workloads. Once the NN model is retrained, it can predict the performance and power of the workload under study in the configuration space and the Pareto Frontier.

Analytical Model (AM): We use the few extracted indicators from the resource utilization traces to build an analytical model. There are two main criteria for this AM model. First, it is able to capture the general trend of performance and power when GPU frequency changes. In other words, it doesn't have to be accurate for all frequency settings. However, we know that it is accurate for the sampled frequencies. Second, it must be simple so that the retraining of the MLP model is fast. To meet these criteria, we adopt the simple, piecewise linear model, and interpolate the performance and power of configurations that are not sampled previously. Note that this AM model becomes more accurate with more points sampled.

Multilayner Perceptron Network (AM+MLP): Multilayner Perceptron Neural networks (MLP) have recently been adopted for performance modeling and prediction. Typically, MLP model training requires a large training dataset of high quality, i.e., many performance events measured by hardware performance counters. Such data collection need intrusive workload profiling and repeated runs of the workloads due to limited number of hardware performance counters available on systems.

As we aim to design a transparent and non-intrusive framework, we are limited to only use profiled performance under a few GPU frequencies in our models. Nevertheless, our framework must be accurate. To address this limitation, we train neural networks using a set of benchmarks and workloads to learn the performance and power trend under GPU frequencies offline. In this work, we trained a model with 3 hidden layers, with 46, 92, 92 neurons on each layer respectively. We

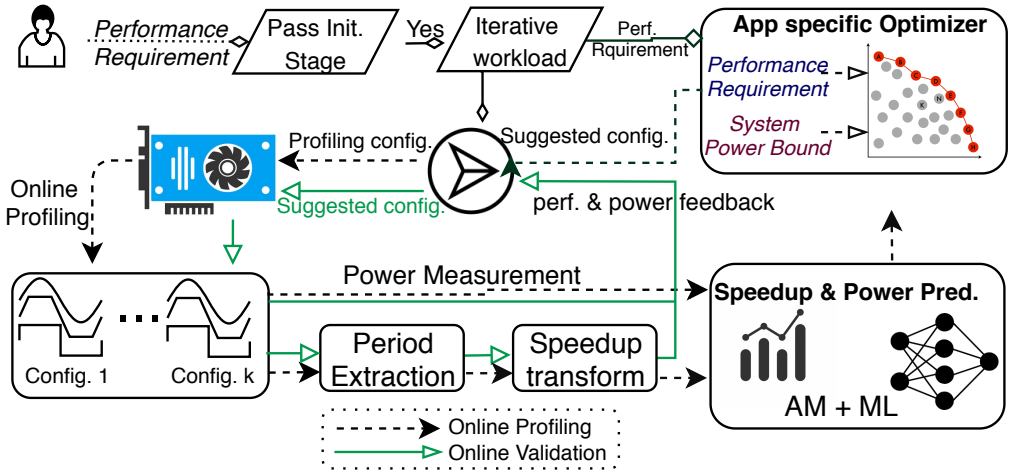


Figure 6.9: The overview of proposed framework.

use Rectified Linear Units (ReLU) as activation functions for the hidden layer, and linear activation function for the output layers respectively.

We further combine the MLP and AM models to improve model accuracy and reduce data requirements. Existing work has already shown that applications tend to follow similar trend of performance and power change when adjusting hardware configurations [76]. Therefore, we re-train and fine-tune the learned MLP model online to adapt to the running workload, which is reflected by the AM model. Specifically, we use the AM as an extra feature for the MLP model and retrain it online. This combination for higher accuracy is known as stacking [51, 30]. This retraining takes negligible time with the simple AM model.

Recurrent Neural Network (AM+RNN): Rather than using MLP models trained from benchmarks and other workloads, we build RNN models to learn power consumption directly from the workloads under study. We adopt the LSTM modeling method. Specifically, our LSTM architecture has 256 steps and takes the resource utilization trace (power, core utilization, memory utilization) as input. It has two hidden layers with 32×16 neurons connected with LSTM layer. In parallel, the AM output are connected with a 46×92 neuron layer. We merge the output from AM and RNN by fully connecting both outputs with a 92 neuron layer for enhanced accuracy.

6.2.5 The Overall Autotuning Framework

Figure 6.9 shows the entire ODPP autotuning framework. After verifying an iterative application has entered its steady state, the agent initiates the online profiling protocol, where the application is assessed under several GPU frequency configurations and a few mapping $f \rightarrow (\texttt{perf}, \texttt{power})$ are extracted. The extracted performance indicators are then fed into the performance/power model for calibration. The calibrated model is then adopted to infer the configurations for Pareto-optimal performance/power combination. All mapping of $f \rightarrow (\texttt{perf}, \texttt{power})$ tuples are stored in a recommendation table. Based on the system management goal (e.g., minimizing energy consumption under QoS constraints, or maximizing performance under a power bound), the Pareto-optimal recommendation table can be different. A configuration from the table that satisfying the goal is then practiced. ODPP keeps tracking the resultant performance and power, verifying whether they meet the QoS requirements or power bound. In case of failure, ODPP selects another recommended configuration for testing. This process continues until an optimal configuration is obtained.

6.3 Experimental Results and Discussion

We evaluate ODPP in this section. We first discuss experiment setup in Subsection-A. We then present the results regarding periodic feature extraction and performance prediction in Subsection-B. After that, we analyze model precision regarding performance and power in Subsection-C and Subsection-D, respectively. Finally, we show how to trade-off between performance QoS constraint and power bound in Subsection-E.

6.3.1 Platform and Applications

Our evaluation platform is a workstation comprising an Intel Haswell 2670-v3 dual-socket processor and an NVIDIA Titan-XP GPU. The GPU supports 4 memory clock levels: 5,705 MHz, 5,505 MHz, 810 MHz and 405 MHz. Under the first three levels, 141 GPU core frequency can be configured; under the last level, only 38 GPU core frequency are supported. We observed in our evaluations that the 810 MHz memory frequency seems not working — it does not make any impact on the performance of the memory. Meanwhile, lowing frequency to 405 MHz will significantly impair the performance where the drag in latency remarkably overshadows the savings from low power in quantifying energy. This is the case in all testings. Finally, as 5,705 MHz is very close

to 5,505 MHz, we only display the evaluation results under 5,705 MHz for the memory part in this section.

We select 8 iterative applications representing typical state-of-the-art workloads in parallel computing, including four deep convolution neural network training applications (*Inception3*, *Inception4*, *ResNet40*, and *ResNet152*), two exascale project (ECP) applications (*CANDLE*[1] and *LAMMPS*) and two ECP proxy applications (*AMG* and *CoMD*). During the training phase of the power/performance prediction models, to have enough training data, we add another 23 benchmark applications from various GPU benchmark suites, including CUDA SDK [78], Rodinia [17], Parboil [96], Tartan [59], etc. Table 6.1 shows the details of evaluated benchmarks.

Table 6.1: The list of benchmarks used to evaluate ODPP.

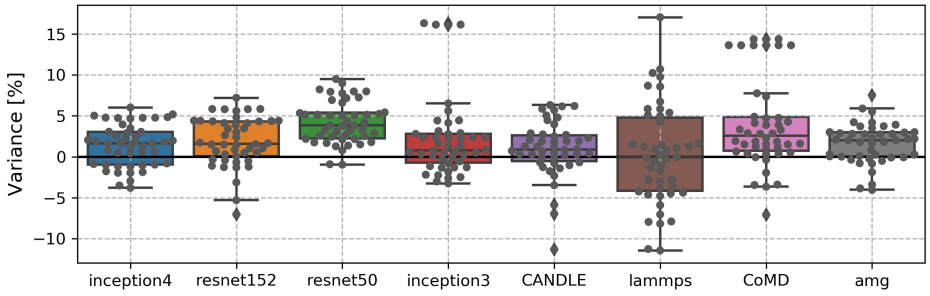
Benchmark	Problem size/Input	Source	Description
Inception3	batch size=64, data=ImageNet	Tensorflow Example	GoogleNet V3
Inception4	batch size=64, data=ImageNet	Tensorflow Example	GoogleNet V4
ResNet40	batch size=64, data=ImageNet	Tensorflow Example	Residual neural network
ResNet152	batch size=64, data=ImageNet	Tensorflow Example	Residual neural network
CANDLE-p1b1	-e=20	ECP	Exascale Deep Learning and Simulation Enabled Precision Medicine for Cancer
LAMMPS	thermo=100, xx=yy=zz=112	ECP	classical molecular dynamics code
AMG	Flan_1565.mtx	ECP proxy	Algebraic Multigrid Benchmark
CoMD	-e -x 100 -y 100 -z 80	ECP proxy	Molecular Dynamics Proxy Application

6.3.2 Accuracy of Speedup Indicator

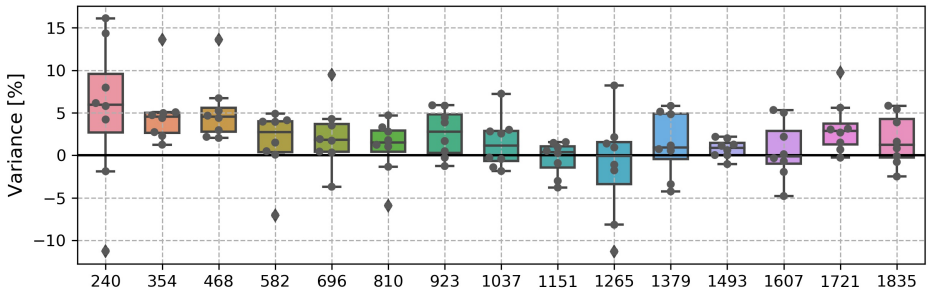
This subsection evaluates our period extraction method and the accuracy of performance prediction based on the extracted period T_{corr_iter} (see Section 6.2-B and C). To verify whether ODPP can precisely predict the performance under different DVFS configurations using performance indicator traces, we draw the predicted normalized speedup values for 46 frequency configurations based on T_{corr_iter} for the 8 applications, and compare them against the applications' real speedups under the same frequency.

To normalize the speedups for displaying purposes, we use the performance measured at the nominal GPU core and memory frequency levels as the baseline. We define relative variance as

[1]We use *P1B1* in the CANDLE benchmark as a representation.



(a) Category by benchmark



(b) Category by frequency (MHz)

Figure 6.10: The error of transform extracted period to normalized speedup.

following to measure accuracy:

$$\texttt{Variance} = \frac{S_{\texttt{measured}} - S_{\texttt{estimated}}}{S_{\texttt{measured}}} \tag{6.7}$$

The box-plots in Figure 6.10 shows the minimum, median and maximum variance (%), and the distribution of the variance between predicted speedups and real measured speedups. As shown, the variance, which is the relative difference between measured and estimated speedup, is different from error and could be negative. The reason is that satisfying users' QoS is usually critical to an HPC center, therefore, predicting speedups lower than real measured for the consequent processing can bring a safer margin than predicting speedups higher than real measured. In other words, when decreasing frequency for saving energy, we have a lower chance in violating QoS constraint. Due to such different consequence on false-positive and false-negative, we show variance in Figure 6.10 (also note it could be negative). As desired, most of the variance are positive.

We also calculate absolute variance (labeled as *error* in the figures) for each test case, where most of them are below 5%. Their average value across all test cases is 3.4%, implying that our speedup prediction using period extracted from the proposed performance indicators is

very accurate. In addition, we have two observations from Figure 6.10: (I) the variance varies across both applications and frequencies. In particular, *LAMMPS* shows the largest variance among others. This is due to relatively more complex execution phase patterns, making period information extraction more difficult. (II) lower core frequencies tend to show relatively higher variance. This is potentially because (a) with lower frequency, the disturbance and overhead from sampling resource utilization & power is different; (b) with lower frequency, more phase-changing details are sampled. Since the major period of an iterative application may include implicit sub-periods, the exhibition of them may interfere the accurate extraction of the major period.

6.3.3 Accuracy of Performance Predictions

This section evaluates the accuracy of performance prediction for unsampled points in the configuration space (see Section 6.2-D). For each application, we predict the speedups under different DVFS configuration, and compare them against the real performance measured. Note, this prediction is different from the prediction in previous subsection in that it is purely inferred by the AM/NN/RNN models, rather than using T_{corr_iter} extracted from the performance indicators.

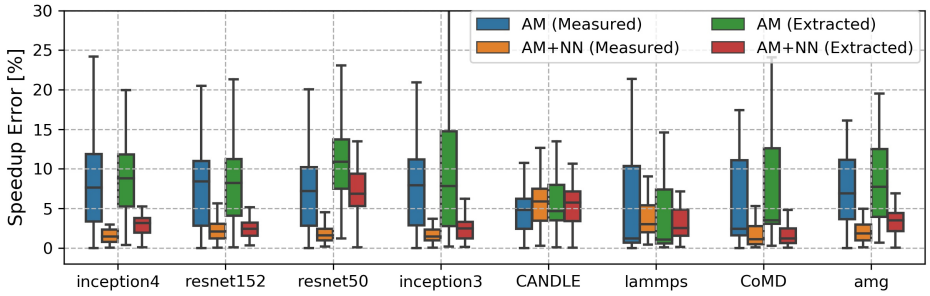


Figure 6.11: Prediction error of speedup. The AM model adopts measured or extracted speedup values as initial point to estimate other configurations with piecewise interpolation.

Figure 6.11 shows the accuracy of prediction of our performance models. AM refers to the piecewise linear interpolation analytical model. AM+NN refers to the MLP model taking AM's output for model initialization.

As can be seen in Figure 6.11, the AM model attains relatively low accuracy in prediction, especially for *CANDLE* and *LAMMPS*. This is because the performance of these two applications do not change in a linear fashion with GPU core frequency. For the training of *Inception* and *Resnet* CNN models, the performance flattens out when the core frequency arrives a certain value

(see Figure 6.7), potentially due to memory saturation. The error of *CoMD* mainly comes from the deviation at lower core frequency (139∼544 MHz), where the performance keeps constant, see Figure 6.3.

As a comparison, the hybrid AM+NN model can improve accuracy by taking advantage of learning from other datasets or models (see Section 6.2-D). With only 3 samples for fine-tuning, it can quickly adapt to the current workload, and reduces prediction error to 3.2% from AM's 9%. It also reduces the percentage of severely miss-predicted (error>10%) cases from 25% to 4%, as compared to AM, significantly lowering the possibility of violating performance QoS. The samples used for model calibration is also much less than AM with a targeted prediction accuracy.

6.3.4 Accuracy of Average Power Predictions

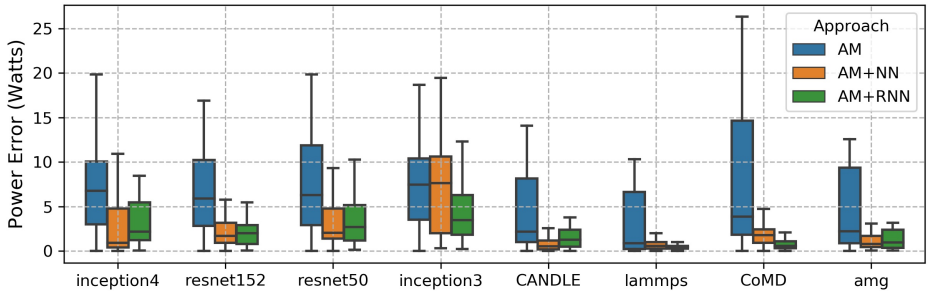


Figure 6.12: Prediction error of power consumption.

Figure 6.12 illustrates the error of prediction for average power consumption across the applications. We use the absolute (rather than normalized) power difference in the figure. As can be seen, AM also shows much higher prediction bias among all the 8 applications. Clearly, the power change under different GPU core frequency is not linear.

Again, the neural networks improve the prediction accuracy dramatically. AM+RNN model achieves slightly better accuracy than AM+NN, particularly for *Inception3*. The reason is that RNN model can learn more underlying characteristics about the running application from the time-series resource utilization traces.

6.3.5 Application Scenarios

We evaluate two performance-power management scenarios:

label	description
Measured	Initial input data measured offline
Extracted	Initial input data extracted online from traces
AM	Performance/power estimated via analytical model
AM + ML	Performance estimated via AM+NN. Power via AM+RNN
Nominal Perf.	Performance/power obtained at the nominal GPU frequency
NVIDIA-SMI	Performance/power obtained with an enforced power bound with `nvidia-smi`

Table 6.2: The description of comparison approach.

6.3.5.1 Minimizing Energy Consumption with Performance Constraints

In cloud computing, the energy bill is the major part of a data center's operating cost. The cloud platform manager expects to increase energy efficiency without violating the quality-of-service (QoS), which is to minimize the energy consumption with performance above certain thresholds. This performance bound is defined as *acceptable speedup* with respect to the performance that can be achieved at the best hardware DVFS configuration.

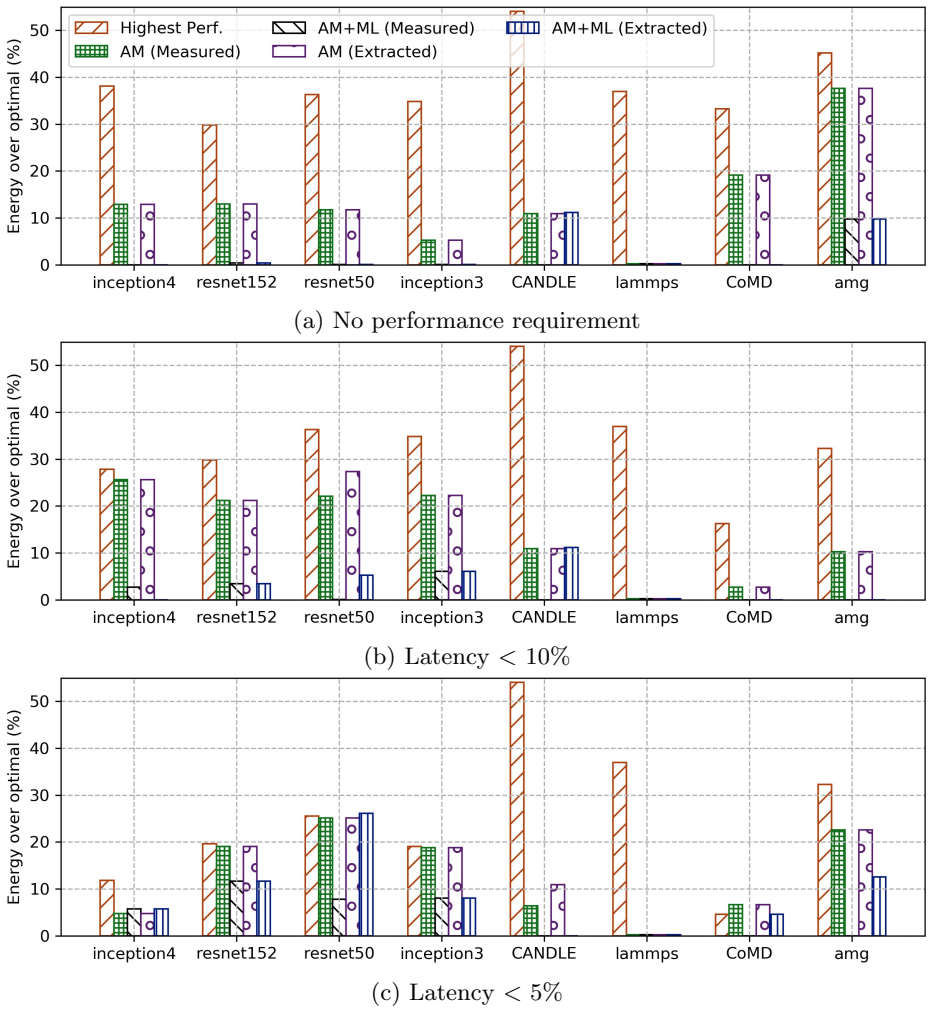
Figure 6.13 shows the predicted optimal configuration's energy efficiency under different approaches. In AM+ML, we predict performance using AM+NN, and predict power using AM+RNN. As can be seen, without any performance requirements, the predicted configuration can save on average 30% (up to 55%) energy consumption compared with full-speed running. For *Inception* and *ResNet* CNN training applications, the AM model reduces acceptable energy consumption, while for *CoMD* and *AMG*, AM also gains more than 20% energy savings.

Figure 6.13b and 6.13c show that for many applications, the configurations predicted by ODPP can deliver the optimal energy efficiency (i.e., the best energy efficiency that can be achieved under the QoS).

Due to the error introduced by period extraction and speedup transformation, the energy based on predicted speedups can be slightly higher than based on real measured speedups. Nevertheless, prediction based on extracted period work pretty well for most of the evaluated applications under various QoS requirements. Specifically, AM+ML (Extracted) incurs 2% deadline misses for *CANDLE* under 10% QoS latency constraint.

6.3.5.2 Maximizing Performance Under A Power Bound

Some supercomputer and data centers are charged by the peak power consumption sampled in a time-frame of several minutes [35]. In such case, the system would benefit from maximizing



(a) No performance requirement

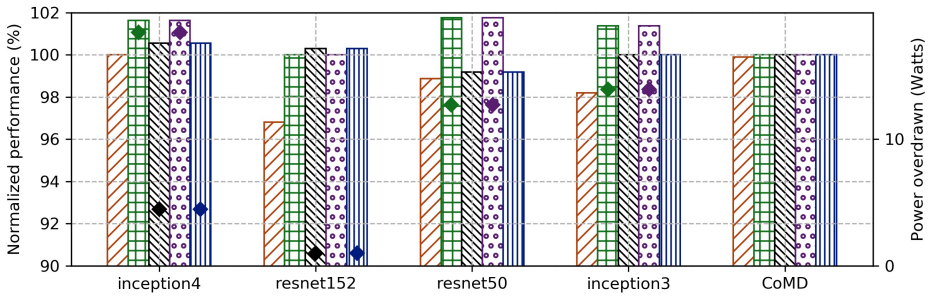
(b) Latency < 10%

(c) Latency < 5%

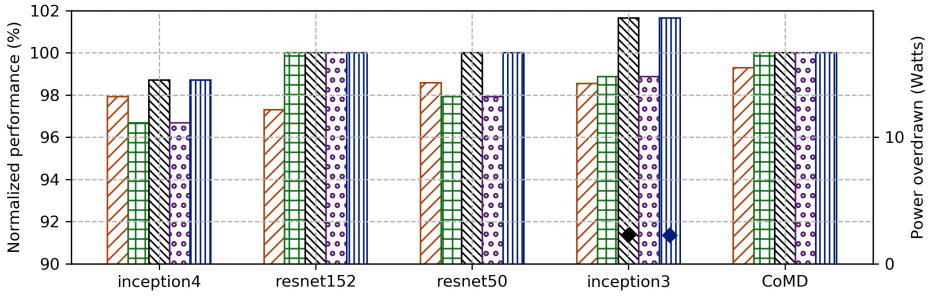
Figure 6.13: Comparison of application minimize energy error under different performance requirements.

performance under a relative lower power bound. We investigate this scenario as to tuning frequency to maximize performance under a certain maximum power bound.

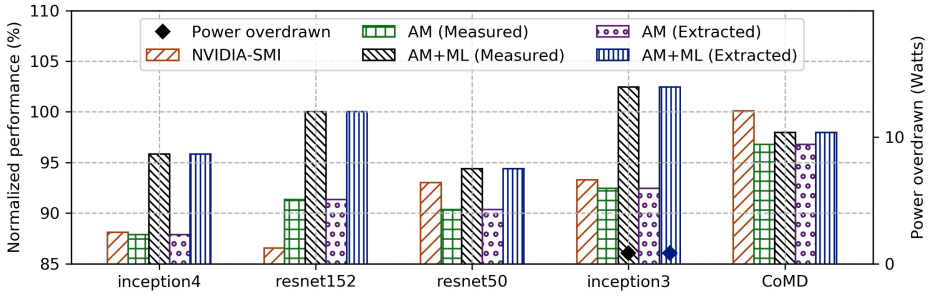
Figure 6.14 shows the estimated optimal configuration's performance under different methods. The performance is normalized to the best performance satisfying the power bounds. We do not show the results of *LAMMPS*, *CANDLE* and *AMG* as they consume less than the lowest power cap 160 Watts on average. As can be seen in the figure, the AM model fails to predict the power very accurately; some applications break the power cap at 220 watts. The AM+ML model also violates the power cap for application *Inception4* and *ResNet152*, but only marginally. We can revise the model prediction during the validation stage. For cases where the power cap is respected, AM+ML



(a) Power Bound = 220 Watts



(b) Power Bound = 190 Watts



(c) Power Bound = 160 Watts

Figure 6.14: Comparison of application performance under different power bounds.

shows the least performance loss compared with the optimal configuration.

Figure 6.14c show that when the specified power cap is low, `nviida-smi` may over-reducing the GPU frequency to enforce the power bound. With 160 watts power-capping level, the performance given by `nvidia-smi` is essentially 10% less than the performance obtained under the optimal configuration. ODPP reports a configuration with 3% performance difference from the optimal, much better than `nvidia-smi`. In addition, the model can predict the extra time required for running an application complying with the power bounds. Such information is important for both system management and the users.

6.4 Summary

This chapter proposes an auto-tuning approach aiming at identifying the optimal GPU frequency settings for iterative workloads on GPUs. It comprises two phases: *detection*, which detects the periodic features from online generated GPU resources utilization traces, and calibrates our performance model; *prediction*, where the features are fed into a hybrid machine learning model to predict performance and power for other DVFS settings, an optimal setting thus can be selected according to desired QoS and power-budget goals. Overall, our major findings and contributions include:

- We present a framework that achieves performance and power objectives by tuning GPU frequency online without code instrumentation or extensive profiling.
- We propose a novel method to extract performance and power indicators from resource utilization traces of iterative workloads. The method monitors application transparently with very low overhead.
- We build a hybrid machine learning model that improves online prediction robustness with small profiling sample sizes, by leveraging the trend indicated by the analytical model and prior knowledge of similar workloads.
- We find that the actual sampling rate of `nvidia-smi` is impacted by current GPU usage and GPU frequency, and propose a strategy to mitigate the impact. Such findings can help researchers revise their energy estimation for GPU application when using `nvidia-smi`.

Evaluations on an NVIDIA Titan-XP GPU show that our ODPP framework can precisely predict the optimal DVFS setting with best performance under power constraints and best energy efficiency satisfying QoS. As iterative workloads account for the major workloads in modern HPC centers, a runtime DVFS tuning framework like ODPP can significantly reduce energy consumption. Therefore, ODPP can be widely applied to both HPC and cloud meeting, by minimizing system power consumption while satisfying customer requested quality ofservice (QoS).

Chapter 7

Conclusions and Future Work

Power bounded for HPC systems is a new area of research, motivated by limiting the power consumption of future exascale systems in the envelop of 20-30 MW. Even though this constraint is not strictly enforced, energy efficiency is still important to operate future HPC system under reasonable cost and deliver a high degree of computational performance. Power bounded computing aims to turn energy saving into application performance with an upper bound of power and energy consumption.

The focus of this dissertation is to maximize application performance as well as system throughput in power bounded HPC systems. We introduced and explored *cross component power coordination* on node level and *cluster level intelligent power coordination*. For node sharing system, we investigated the impact of power cap and designed and implemented CAPS to maximize system throughput for such node sharing systems. Finally, to meet the QoS without offline profiling and code intrusion, we proposed *online dynamic power coordination* to maximize energy efficiency and satisfy QoS on heterogeneous architectures. We summarize the contributions of this dissertation below.

7.1 Research Summary

As HPC systems are increasingly bounded by power budget, it is clear that future HPC systems and software must cope with these power bounds. The key idea is how to distribute hardware resource and power optimally to achieve maximize performance under a power bound. In this work,

we first study the relationship between application characteristic and power allocation. Our studies reveal there exist categorical patterns in the dynamics between power allocation and application performance. Such patterns are corresponding to application characteristic, including: scalability, memory intensity, core affinity, and critical power levels. Based on these insights, we propose and evaluate an application characteristic aware power coordination method that seeks an optimal core allocation and power coordination that maximizes application performance under given power budgets. The presented power coordination method delivers near-optimal performance for applications with various characteristics and outperforms default strategies significantly especially for memory intensive and medium scalable applications. The cross component power coordination can be easily extended to cluster level or heterogeneous systems.

To enable power bounded computing on the cluster level, we first focused on power bounded computing at the node level by cross component power coordination. We designed and implemented CLIP, a hierarchical multi-dimensional power aware allocation framework. CLIP built a performance model to determine the optimal number of participating compute nodes and components, and their power distributions for given applications based on application hardware events and high level scalability. CLIP manages power coordination at both cluster level and node level. At the cluster level, CLIP determines the number of nodes according to optimal hardware and power configuration of the application at node level. At the node level, CLIP determines the number of activated cores, core affinity and power coordination between components based on application characteristics. Experimental results on a Haswell-based computer cluster show that the proposed scheduler outperforms compared methods by over 20% on average for various power budgets.

To address power bounded computing for node sharing systems, we designed and implemented CAPS to schedule jobs and manage resources in a find-grained way. CAPS collocates jobs that are complementary when power is limited, and distributes the available power to nodes and components to minimize their interference. CAPS co-schedules jobs are both power complementary and memory intensity complementary. CAPS estimates the power demands of collocated jobs with a neural network model. With accurate power demands prediction, collocated jobs would not either waste power budgets or be heavily impacted by power bounds. CAPS also avoid heavily memory intensive applications to co-run on the same node to address memory contention by building performance models to predict workload interference due to resource contention and power limitations. Besides, CAPS takes the interference of power capping to memory intention into consideration, and

shifts power budget from nodes and components to increase system throughput. CAPS significantly increases system performance, power utilization, and energy efficiency on an experimental cluster.

On GPU architecture, we focus on a common type of workloads: iterative workloads. Iterative workloads illustrate periodical resource utilization variation, which offer the possibility to monitor performance in real time without code intrusion. ODPP exploit such iterative or periodic features of conventional HPC and emerging machine learning workloads. ODPP extracts the performance and power indicators for applications from easily obtained GPU resource utilization profiles in a short episode. The extraction method uses Fourier transform and incurs minimal cost. ODPP further automatically constructs an accurate model that infers from the indicators how the application's performance and power vary with GPU core and memory frequencies. Aided with the model, for both seen and unseen applications, ODPP can quickly determine the most appropriate DVFS configuration for their execution. The evaluation was done on an NVIDIA GPU using multiple exascale computing (ECP) and deep learning applications. Evaluation results show that ODPP can improve energy efficiency by over 30% under different QoS and improve performance by more than 8% under different power bounds.

7.2 Research Insights

This dissertation provides multiple insights in system scale power management to address the power challenge for exascale supercomputing systems. We provide a novelty technology–power bounded computing, which maximally transfer power budget to application performance and system throughput. We implement power bounded computing on different levels and apply for systems with different purposes. The contribution of this dissertation can be summarized as follows:

- The dissertation validates the effectiveness of power bounded computing for contribution to exascale system energy efficiency. Power bounded computing is an additional technology needed to turn power budget into additional performance and system throughput.
- We propose multiple power bounded computing techniques to improve application performance and system throughput for both node level and cluster level, both job monopoly and node sharing systems, both homogeneous and heterogeneous systems. The evaluation on different systems and architectures validates the effectiveness and generality of power bounded computing.

- We evaluate power bounded computing with applications which have different kinds of characteristics, including computational intensive and memory intensive workloads, traditional HPC workloads and emerging artificial intelligence workloads. We prove that power bounded computing works for both current popular workloads and future critical applications.
- Our work transparently and dynamically trades off between application performance and system power consumption without code intrusion or offline profiling. The technologies proposed in this dissertation can be extended to cloud computing and edge computing for both performance requirement and energy efficiency.

The exascale supercomputer is expected to consume a large amount of power. The CPU, memory, interconnect and platform technologies shall evolve to bring the power consumption closer to the goal of 20 Megawatts. Power bounded computing focuses on system level, improving system energy efficiency with revolutionary hardware and fine-grained management. Power bounded computing has the potential to provide even further power benefits to help meet the exascale energy goals.

7.3 Future Work

This dissertation has laid the groundwork for power bounded research as we move toward exascale supercomputing. Long term future directions can seek to develop the ideas around power bounded computing for emerging hardware, power stable computing, self-learning resource management, and the trade-off between power, performance and resilience. In each of the following future directions, building and deploying large scale systems for future HPC and machine learning applications will be the focus. The following research topics are core to the future of exascale computing; each has significant, very interesting and challenging unsolved problems.

7.3.1 Power Bounded Computing for Emerging Hardware

Over the past few decades, the major computational power components of supercomputers have evolved from CPU to various processor components, including: Graphic Processing Units(GPU), Field-Programmable Gate Array(FPGA), etc. At the same time, the memory technology has also evolved significantly to catch up the speed of processor development. The non-volatile memory is the most popular and had been integrated on current HPC systems. Multiple hierarchy

memory technologies are also introduced to increase the memory bandwidth. For example, both Intel Knights Landing and NVIDIA Volta V100 processors integrated HBM (High Bandwidth Memory) on the chip, at the same time, the system typically deployed large capacity DDR4.

The increasing heterogeneity on both processor and memory sides improves the difficulties and enlarges the space for power bounded computing. On one hand, the advance processors and memory increase application performance and power efficiency, and enable power bounded computing has wider space to coordinate hardware resource and power to achieve better application performance. On the other hand, the impact of power cap to the advance hardware has not been studied thoroughly. Further, power bounded computing needs to identify and exploit the interaction influence between application performance and power cap. The future power bounded computing needs to accommodate for these advances and multiple hierarchical hardware and develop multi-dimensional application aware power coordination algorithms to improve application performance and system energy efficiency. Further work can extend ODPP in Chapter 6 to deal with multi-GPU system and large scale heterogeneous systems.

7.3.2 Power Stable Computing

Power variation on HPC systems has been long overlooked. With the development of hardware (eg. power state change in CPU and memory), the power variation in a data center increase significantly from historically five percent to more than eighty percent [31, 95]. Power variation in HPC systems generates several types of problems:

- Power variation has caused redundancy loss of cooling service and power delivery infrastructures [103, 101]. The auxiliary support equipments need to guarantee the safety of data center even it operates at high power demands.

- Power variation between nodes leads to regional overheat and increases the possibility of system fault [79, 12]. Nodes with high computational demands jobs increase the power consumption on regional cabinets, and may generate more heat. Thus, power variation boots the chance of system fault which is related with regional temperature.

- Power variation increases the difficulty for power supply institutions [54, 31]. Power grid bears much more burden, because of the sub unitary power factor of data center. Power grid operators even consider to charge penalties from data center owner [21]. For example, NSCA

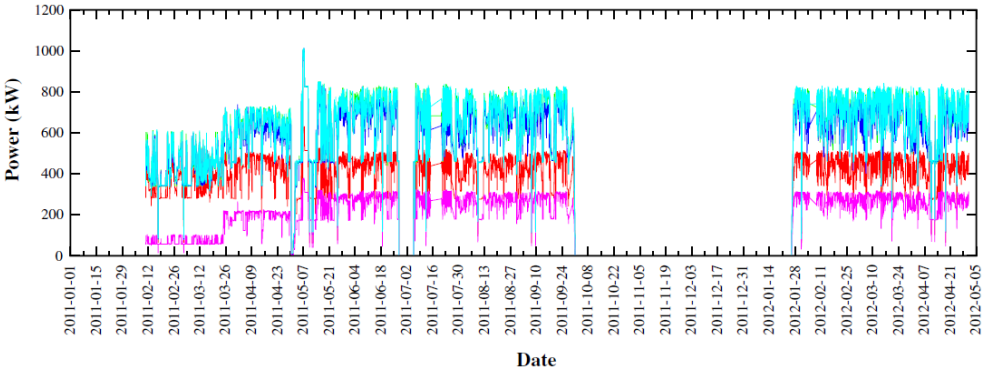


Figure 7.1: Per-Switchboard power data for each of Cielo's five switchboards. [80]

and LLNL have signed the contract with power supply institution by charging power at peak power consumption [22].

Unfortunately, modern hardware development deteriorates power variation in data centers. Currently, both CPU and memory supports multiple power states. The components variety increases system performance, but also deteriorates the power variation. Multi-cores and many-cores enlarge power variation on the CPU, as the applications operates at different concurrency and gains different speedups. Besides, the heterogeneity among HPC data center rises up power consumption unbalance. In fact, the power consumption of HPC system varies from 200 kW to 800 kW [80]. Figure 7.1 shows the power variance of Trinity data center in Los Alamos National Laboratory from January 2011 to May 2012. With the uprise demand for computing resource, the power variation will boost and make power supply institution harder to management.

Power bounded or power-aware techniques are not enough to solve power variation issues. Power bounded computing techniques [44, 68] are able to improve performance under a power envelop, but they overlooked the potential power budget underutilization of executing a power judicious job. Power sharing between jobs can decrease power variation [34] with the risk to increase regional overheat. Co-location jobs is capable to drop down power variation if cooperated with coordinated power management strategies [107]. The integration of power bounded, power sharing and job co-running techniques offers an practical solution to address power variation. In the future, power bounded computing needs to integrate with other techniques to ensure HPC system running at a reasonable power range without unexpected power fluctuation.

7.3.3 Self-learning Resource Management

One of the key observations from prior research with ODPP is that the scheduling policies can be determined dynamically and online. In a large scale system, the QoS of jobs may change while running, the system power consumption could change due to job enter and exit, and the power cost unit could vary because of the nature of power grid. The scheduling policy must adapt dynamically to fit the demands or environment changes.

A self-learning resource manager has the ability to learn and identify the best scheduling policy based on historical data and some criteria (such as application scalability, critical power levels, etc). A self-learning resource manager must also adopt the control theory to handle the dynamical job requirement and environment changes. Reinforcement machine learning provides a feasible way to utilize relevant data and knowledge and determine what policies should be deployed in which scenarios.

7.3.4 Trade-off between Power, Performance and Resilience

System reliability has been identified as one of the most important challenges faced by high performance computing (HPC) [13]. To achieve exascale computing, the system will integrate larger number of nodes and hundreds of cores per node. The large platform size makes the failures are unable to avoid during the application execution in exascale era. The complexity of exascale system rises up significantly due to much more heterogeneous components and environment. Processors encounter more soft-errors and hardware errors to apply shrinking process technology for heterogeneous computing [14, 48]. Analysis of operational logs indicates that the mean time between failure (MTBF) will be much shorter in the exascale systems. Therefore, HPC systems have to deploy effective tools and solutions to address resilience issues [48].

Energy and power consumption is also one of the top challenge for exascale computing [64]. The energy cost has taken the largest proportion of both the HPC and commercial data center operation cost. As the increasing demands of higher computational capability, the energy bill also increase significantly per year. The HPC has to shift from improving performance without considering power consumption to execute application with higher energy efficiency. The future computer system may have to trade-off between performance and energy consumption according to the job priority, energy price and available power supply.

Resilience and energy are both important problems that have to be addressed for next generation exascale computing system [91]. As expected, both resilience and energy saving techniques will impact workload performance. Previous works have proposed several energy models to estimate the energy consumption of different resilience technologies [73, 33, 72].

We had investigated the impact of multilevel checkpoint on THETA production system deployed at Argonne National Laboratory. The THETA system's massively parallel many-core architecture and heterogeneous memory architecture bring new opportunities and challenges for scientific applications. The Dragonfly network enables the system share I/O with communication. We identify that the checkpointing on each level cause different performance and energy overhead to application. Power bounded computing must consider such features to avoid increasing failure chance and therefore higher energy consumption.

Bibliography

[1] Palmetto cluster documentation. https://www.palmetto.clemson.edu/palmetto/.

[2] Tesla k20 gpu active accelerator board specification. https://www.nvidia.com/content/PDF/kepler/tesla-k20-active-bd-06499-001-v03.pdf.

[3] Top500 list. https://www.top500.org/, 2017. Accessed: 2017-08-06.

[4] Cori queues and policies. http://www.nersc.gov/users/computational-systems/cori/running-jobs/queues-and-policies/, 2018.

[5] B. Acun, A. Buyuktosunoglu, E. K. Lee, and Y. Park. Power aware heterogeneous node assembly. In *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 715–727, Feb 2019.

[6] ANL. Aurora announcement. https://www.anl.gov/aurora-announcement.

[7] David H Bailey, Eric Barszcz, John T Barton, et al. The NAS parallel benchmarks. *International Journal of High Performance Computing Applications*, 5(3):63–73, 1991.

[8] L. A. Barroso, U. Hölzle, P. Ranganathan, and M. Martonosi. *The Datacenter as a Computer: Designing Warehouse-Scale Machines, Third Edition*. 2018.

[9] Anton Beloglazov, Jemal Abawajy, and Rajkumar Buyya. Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. *Future Generation Computer Systems*, 28(5):755 – 768, 2012. Special Section: Energy efficiency in large-scale distributed systems.

[10] S. Blagodurov, S.and Zhuravlev and A. Fedorova. Contention-aware scheduling on multicore systems. *ACM Trans. Comput. Syst.*, 28(4):8:1–8:45, December 2010.

[11] J. Breitbart, J. Weidendorfer, and C. Trinitis. Case study on co-scheduling for hpc applications. In *2015 44th International Conference on Parallel Processing Workshops*, pages 277–285, Sept 2015.

[12] Thang Cao, Wei Huang, Yuan He, and Masaaki Kondo. Cooling-aware job scheduling and node allocation for overprovisioned hpc systems. In *2017 IEEE International Symposium on Parallel Distributed Processing (IPDPS)*, pages 728–737. IEEE, 2017.

[13] Franck Cappello, Geist Al, William Gropp, Sanjay Kale, Bill Kramer, and Marc Snir. Toward exascale resilience: 2014 update. *Supercomput. Front. Innov.: Int. J.*, 1(1):5–28, April 2014.

[14] Franck Cappello, Al Geist, Bill Gropp, Laxmikant Kale, Bill Kramer, and Marc Snir. Toward exascale resilience. *Int. J. High Perform. Comput. Appl.*, 23(4):374–388, November 2009.

[15] Dimitrios Chasapis, Marc Casas, Miquel Moretó, Martin Schulz, Eduard Ayguadé, Jesus Labarta, and Mateo Valero. Runtime-guided mitigation of manufacturing variability in power-constrained multi-socket numa nodes. In *Proceedings of the 2016 International Conference on Supercomputing*, ICS '16, pages 5:1–5:12, New York, NY, USA, 2016. ACM.

[16] Vincent Chau, Xiaowen Chu, Hai Liu, and Yiu-Wing Leung. Energy Efficient Job Scheduling with DVFS for CPU-GPU Heterogeneous Systems. In *Proceedings of the Eighth International Conference on Future Energy Systems - e-Energy '17*, pages 1–11. ACM Press.

[17] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S. Lee, and K. Skadron. Rodinia: A benchmark suite for heterogeneous computing. In *2009 IEEE International Symposium on Workload Characterization (IISWC)*, pages 44–54, Oct 2009.

[18] Ming Chen, Xiaorui Wang, and Xue Li. Coordinating processor and main memory for efficientserver power control. In *Proceedings of the International Conference on Supercomputing*, ICS '11, pages 130–140, New York, NY, USA, 2011. ACM.

[19] Ming Chen, Xiaorui Wang, and Xue Li. Coordinating Processor and Main Memory for Efficientserver Power Control. In *Proceedings of the international conference on Supercomputing*. ACM, 2011.

[20] J. Choi, M. Dukhan, X. Liu, and R. Vuduc. Algorithmic Time, Energy, and Power on Candidate HPC Compute Building Blocks. In *Parallel and Distributed Processing Symposium, 2014 IEEE 28th International*, pages 447–457, May 2014.

[21] Tudor Cioara, Ionut Anghel, Ioan Salomie, Marcel Antal, Massimo Bertoncini, and Diego Arnone. Optimizing the power factor of data centers connected to the smart grid. In *Proceedings of the 5th International Workshop on Energy Efficient Data Centres*, E2DC '16, pages 3:1–3:6, New York, NY, USA, 2016. ACM.

[22] Anders Clausen, Gregory Koenig, Sonja Klingert, Girish Ghatikar, Peter M. Schwartz, and Natalie Bates. An analysis of contracts and relationships between supercomputing centers and electricity service providers. In *Proceedings of the 48th International Conference on Parallel Processing: Workshops*, ICPP 2019, New York, NY, USA, 2019. Association for Computing Machinery.

[23] Ryan Cochran, Can Hankendi, Ayse K. Coskun, and Sherief Reda. Pack & cap: Adaptive dvfs and thread packing under power caps. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-44, pages 175–185, New York, NY, USA, 2011. ACM.

[24] Daniel Dauwe, Ryan Friese, Sudeep Pasricha, and et al. Modeling the Effects on Power and Performance from Memory Interference of Co-located Applications in Multicore Systems. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*, 2014.

[25] H. David, E. Gorbatov, U. R. Hanebutte, R. Khanna, and C. Le. Rapl: Memory power estimation and capping. pages 189–194, Aug 2010.

[26] Howard David, Chris Fallin, Eugene Gorbatov, Ulf R. Hanebutte, and Onur Mutlu. Memory power management via dynamic voltage/frequency scaling. In *Proceedings of the 8th ACM International Conference on Autonomic Computing*, ICAC '11, pages 31–40, New York, NY, USA, 2011. ACM.

[27] A. De Blanche and T. Lundqvist. Node sharing for increased throughput and shorter runtimes : an industrial co-scheduling case study. In *Proceedings of the 3rd Workshop on Co-Scheduling of HPC Applications (COSH 2018) : Held together with HiPEAC 2018*, pages 15–20, 2018.

[28] Daniele De Sensi, Massimo Torquati, and Marco Danelutto. A reconfiguration algorithm for power-aware parallel applications. *ACM Trans. Archit. Code Optim.*, 13(4):43:1–43:25, December 2016.

[29] Qingyuan Deng, David Meisner, Abhishek Bhattacharjee, Thomas F. Wenisch, and Ricardo Bianchini. Coscale: Coordinating cpu and memory system dvfs in server systems. In *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-45, pages 143–154, Washington, DC, USA, 2012. IEEE Computer Society.

[30] Diego Didona, Francesco Quaglia, Paolo Romano, and Ennio Torre. Enhancing performance prediction robustness by combining analytical modeling and machine learning. In *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering*, ICPE '15, pages 145–156, New York, NY, USA, 2015. ACM.

[31] Patric Donovan. An overlooked problem: Dynamic power variations. `http://www.datacenterknowledge.com/`.

[32] T. Dwyer, A. Fedorova, S. Blagodurov, M. Roth, F. Gaud, and J. Pei. A practical method for estimating performance degradation on multicore processors, and its application to hpc workloads. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '12, pages 83:1–83:11, Los Alamitos, CA, USA, 2012. IEEE Computer Society Press.

[33] N. El-Sayed and B. Schroeder. To checkpoint or not to checkpoint: Understanding energy-performance-i/o tradeoffs in hpc checkpointing. In *2014 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 93–102, Sept 2014.

[34] D. A. Ellsworth, A. D. Malony, B. Rountree, and M. Schulz. Dynamic Power Sharing for Higher Job Throughput. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '15. ACM, 2015.

[35] Ryanand Elmore, Kenny Gruchalla, Caleb Phillips, Avi Purkayastha, and Nick Wunder. Analysis of Application Power and Schedule Composition in a High Performance Computing Environment.

[36] M. Etinski, J. Corbalan, J. Labarta, and M. Valero. Parallel job scheduling for power constrained hpc systems. *Parallel Comput.*, 38(12):615–630, December 2012.

[37] S. Eyerman and L. Eeckhout. System-level performance metrics for multiprogram workloads. *IEEE Micro*, 28(3):42–53, May 2008.

[38] Oak Ridge Leadership Computing Facility. Summit user guide. `https://www.olcf.ornl.gov/for-users/system-user-guides/summit/summit-user-guide/#running-jobs/`.

[39] Kaijie Fan, Biagio Cosenza, and Ben Juurlink. Predictable GPUs frequency scaling for energy and performance. In *Proceedings of the 48th International Conference on Parallel Processing - ICPP 2019*, pages 1–10. ACM Press.

[40] R. Ge and K. W. Cameron. Power-aware speedup. In *2007 IEEE International Parallel and Distributed Processing Symposium*, pages 1–10, 2007.

[41] R. Ge, X. Feng, W. Feng, and K. W. Cameron. Cpu miser: A performance-directed, run-time system for power-aware clusters. In *2007 International Conference on Parallel Processing (ICPP 2007)*, pages 18–18, Sep. 2007.

[42] R. Ge, X. Feng, Y. He, and P. Zou. The case for cross-component power coordination on power bounded systems. In *2016 45th International Conference on Parallel Processing (ICPP)*, pages 516–525, Aug 2016.

[43] R. Ge, R. Vogt, J. Majumder, A. Alam, M. Burtscher, and Z. Zong. Effects of dynamic voltage and frequency scaling on a k20 gpu. In *2013 42nd International Conference on Parallel Processing*, pages 826–833, Oct 2013.

[44] R. Ge, P. Zou, and X. Feng. Application-aware power coordination on power bounded numa multicore systems. In *2017 46th International Conference on Parallel Processing (ICPP)*, pages 591–600, Aug 2017.

[45] Neha Gholkar, Frank Mueller, and Barry Rountree. Power Tuning HPC Jobs on Power-Constrained Systems. In *Proceedings of the 2016 International Conference on Parallel Architectures and Compilation - PACT '16*, pages 179–191, Haifa, Israel, 2016. ACM Press.

[46] Neha Gholkar, Frank Mueller, and Barry Rountree. Power tuning hpc jobs on power-constrained systems. In *Proceedings of the 2016 International Conference on Parallel Architectures and Compilation*, PACT '16, pages 179–191, New York, NY, USA, 2016. ACM.

[47] J. Guerreiro, A. Ilic, N. Roma, and P. Tomas. GPGPU Power Modeling for Multi-domain Voltage-Frequency Scaling. pages 789–800, February 2018.

[48] Saurabh Gupta, Tirthak Patel, Christian Engelmann, and Devesh Tiwari. Failures in large scale systems: Long-term measurement, analysis, and implications. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '17, pages 44:1–44:12, New York, NY, USA, 2017. ACM.

[49] H. Hanson, W. Felter, W. Huang, C. Lefurgy, K. Rajamani, F. Rawson, and G. Silva. Processor-Memory Power Shifting for Multi-Core Systems. In *4th Workshop on Energy Efficient Design*, 2012.

[50] Sunpyo Hong and Hyesoon Kim. An Integrated GPU Power and Performance Model. In *Proceedings of the 37th Annual International Symposium on Computer Architecture*, ISCA '10, pages 280–289. ACM.

[51] H. Ibeid, S. Meng, O. Dobon, L. Olson, and W. Gropp. Learning with analytical models. In *2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 778–786, May 2019.

[52] Yuichi Inadomi, Tapasya Patki, Koji Inoue, Mutsumi Aoyagi, Barry Rountree, Martin Schulz, David Lowenthal, Yasutaka Wada, Keiichiro Fukazawa, Masatsugu Ueda, Masaaki Kondo, and Ikuo Miyoshi. Analyzing and mitigating the impact of manufacturing variability in power-constrained supercomputing. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '15, pages 78:1–78:12, New York, NY, USA, 2015. ACM.

[53] David Jackson, Quinn Snell, and Mark Clement. Core Algorithms of the Maui Scheduler. In Dror G. Feitelson and Larry Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, pages 87–102. Springer Berlin Heidelberg.

[54] Vahid Jalili-Marandi, Zhiyin Zhou, and Venkata Dinavahi. Large-scale transient stability simulation of electrical power systems on parallel gpus. In *Power and Energy Society General Meeting, 2012 IEEE*, pages 1–11. IEEE, 2012.

[55] K. Kasichayanula, D. Terpstra, P. Luszczek, S. Tomov, S. Moore, and G. D. Peterson. Power Aware Computing on GPUs. In *Application Accelerators in High Performance Computing (SAAHPC), 2012 Symposium on*, pages 64–73, July 2012.

[56] J. Kelley, C. Stewart, D. Tiwari, and S. Gupta. Adaptive Power Profiling for Many-Core HPC Architectures. In *2016 IEEE International Conference on Autonomic Computing (ICAC)*, pages 179–188, July 2016.

[57] T. Komoda, S. Hayashi, T. Nakada, S. Miwa, and H. Nakamura. Power Capping of CPU-GPU Heterogeneous Systems through Coordinating DVFS and Task Mapping. In *31st International Conference on Computer Design*, pages 349–356, Oct 2013.

[58] Young Choon Lee and Albert Y. Zomaya. Energy efficient utilization of resources in cloud computing systems. *The Journal of Supercomputing*, 60(2):268–280, May 2012.

[59] Ang Li, Shuaiwen Leon Song, Jieyang Chen, Xu Liu, Nathan Tallent, and Kevin Barker. Tartan: evaluating modern gpu interconnect via a multi-gpu benchmark suite. In *2018 IEEE International Symposium on Workload Characterization (IISWC)*, pages 191–202. IEEE, 2018.

[60] J. Li and J. F. Martinez. Dynamic power-performance adaptation of parallel computation on chip multiprocessors. In *The Twelfth International Symposium on High-Performance Computer Architecture, 2006.*, pages 77–87, Feb 2006.

[61] J. Li, J. F. Martinez, and M. C. Huang. The thrifty barrier: energy-aware synchronization in shared-memory multiprocessors. In *10th International Symposium on High Performance Computer Architecture (HPCA'04)*, pages 14–23, Feb 2004.

[62] Xiaodong Li, Ritu Gupta, Sarita V. Adve, and Yuanyuan Zhou. Cross-component energy management: Joint adaptation of processor and memory. *ACM Trans. Archit. Code Optim.*, 4(3), September 2007.

[63] LLNL. DOE/NNSA, Lab announce partnership with Cray to develop NNSA's first exascale supercomputer. `https://www.llnl.gov//news`.

[64] R Lucas, J Ang, K Bergman, S Borkar, W Carlson, L Carrington, G Chiu, R Colwell, W Dally, J Dongarra, et al. Top ten exascale research challenges. *DOE ASCAC subcommittee report*, pages 1–86, 2014.

[65] Piotr R Luszczek, David H Bailey, Jack J Dongarra, Jeremy Kepner, Robert F Lucas, Rolf Rabenseifner, and Daisuke Takahashi. The HPC Challenge (HPCC) benchmark suite. In *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, page 213, 2006.

[66] A. Majumdar, L. Piga, I. Paul, J. L. Greathouse, W. Huang, and D. H. Albonesi. Dynamic GPGPU Power Management Using Adaptive Model Predictive Control. In *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 613–624, February 2017.

[67] A. Marathe, P. E. Bailey, D. K. Lowenthal, B. Rountree, M. Schulz, and B. R. de Supinski. A run-time system for power-constrained hpc applications. In J. M. Kunkel and T. Ludwig, editors, *High Performance Computing*, pages 394–408, Cham, 2015. Springer International Publishing.

[68] Aniruddha Marathe, Rushil Anirudh, Nikhil Jain, Abhinav Bhatele, Jayaraman Thiagarajan, Bhavya Kailkhura, Jae-Seung Yeom, Barry Rountree, and Todd Gamblin. Performance modeling under resource constraints using deep transfer learning. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '17, pages 31:1–31:12, New York, NY, USA, 2017. ACM.

[69] X. Mei, X. Chu, H. Liu, Y. Leung, and Z. Li. Energy efficient real-time task scheduling on CPU-GPU hybrid clusters. In *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*, pages 1–9.

[70] Xinxin Mei, Ling Sing Yung, Kaiyong Zhao, and Xiaowen Chu. A measurement study of gpu dvfs on energy conservation. In *Proceedings of the Workshop on Power-Aware Computing and Systems*, HotPower '13, pages 10:1–10:5, New York, NY, USA, 2013. ACM.

[71] P. Messina. Update on the exascale computing project (ecp). HPC User Forum, 2017.

[72] B. Mills, T. Znati, R. Melhem, K. B. Ferreira, and R. E. Grant. Energy consumption of resilience mechanisms in large scale systems. In *2014 22nd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, pages 528–535, Feb 2014.

[73] Bryan Mills, Ryan E. Grant, Kurt B. Ferreira, and Rolf Riesen. Evaluating energy savings for checkpoint/restart. In *Proceedings of the 1st International Workshop on Energy Efficient Supercomputing*, E2SC '13, pages 6:1–6:8, New York, NY, USA, 2013. ACM.

[74] N. Mishra, J. D. Lafferty, and H. Hoffmann. Esp: A machine learning approach to predicting application interference. In *2017 IEEE International Conference on Autonomic Computing (ICAC)*, pages 125–134, July 2017.

[75] Nikita Mishra, Connor Imes, John D. Lafferty, and Henry Hoffmann. Caloree: Learning control for predictable latency and low energy. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '18, pages 184–198, New York, NY, USA, 2018. ACM.

[76] Nikita Mishra, Huazhe Zhang, John D. Lafferty, and Henry Hoffmann. A probabilistic graphical model-based approach for minimizing energy under performance constraints. In *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '15, pages 267–281, New York, NY, USA, 2015. ACM.

[77] Sebstian Nussbaum. Amd Trinity Fusion APU. In *Proceedings of the Hot Chips: A Symposium on High Performance Chips*, 2012.

[78] NVIDIA. Gpu computing sdk. `https://developer.nvidia.com/gpucomputing-sdk/`.

[79] Ali Pahlavan, Mahmoud Momtazpour, and Maziar Goudarzi. Data center power reduction by heuristic variation-aware server placement and chassis consolidation. In *Computer Architecture and Digital Systems (CADS), 2012 16th CSI International Symposium on*, pages 150–155. IEEE, 2012.

[80] Scott Pakin, Curtis Storlie, Michael Lang, Robert E Fields, Eloy E Romero, Craig Idler, Sarah Michalak, Hugh Greenberg, Josip Loncaric, Randal Rheinheimer, et al. Power usage of production supercomputers and production workloads. *Concurrency and Computation: Practice and Experience*, 28(2):274–290, 2016.

[81] Tirthak Patel and Devesh Tiwari. Perq: Fair and efficient power management of power-constrained large-scale computing systems. In *Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '19, page 171–182, New York, NY, USA, 2019. Association for Computing Machinery.

[82] T. Patki, D. K. Lowenthal, A. Sasidharan, M. Maiterth, B. L. Rountree, M. Schulz, and B. R. de Supinski. Practical resource management in power-constrained, high performance computing. In *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '15, pages 121–132, New York, NY, USA, 2015. ACM.

[83] Tapasya Patki. The case for hardware overprovisioned supercomputers. 2015.

[84] Tapasya Patki, David K Lowenthal, Barry Rountree, Martin Schulz, and Bronis R De Supinski. Exploring hardware overprovisioning in power-constrained, high performance computing. In *Proceedings of the 27th international ACM conference on International conference on supercomputing*, pages 173–182. ACM, 2013.

[85] Tapasya Patki, David K. Lowenthal, Barry L. Rountree, Martin Schulz, and Bronis R. de Supinski. Economic Viability of Hardware Overprovisioning in Power-constrained High Performance Computing. In *Proceedings of the 4th International Workshop on Energy Efficient Supercomputing*, E2SC '16, pages 8–15, Piscataway, NJ, USA, 2016. IEEE Press.

[86] Indrani Paul, Wei Huang, Manish Arora, and Sudhakar Yalamanchili. Harmonia: Balancing Compute and Memory Power in High-performance GPUs. In *Proceedings of the 42Nd Annual International Symposium on Computer Architecture*, ISCA '15, pages 54–65. ACM.

[87] A. Purkayastha, S. Hammond, R. Nagappan, and M. A. ex-Intel. Holistic approaches to hpc power and workflow management*. In *2018 Ninth International Green and Sustainable Computing Conference (IGSC)*, pages 1–8, Oct 2018.

[88] B. Rountree, D. H. Ahn, B. R. de Supinski, D. K. Lowenthal, and M. Schulz. Beyond dvfs: A first look at performance under a hardware-enforced power bound. In *2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops PhD Forum*, pages 947–953, May 2012.

[89] Barry Rountree, David K. Lownenthal, Bronis R. de Supinski, Martin Schulz, Vincent W. Freeh, and Tyler Bletsch. Adagio: Making dvs practical for complex hpc applications. In *Proceedings of the 23rd International Conference on Supercomputing*, ICS '09, pages 460–469, New York, NY, USA, 2009. ACM.

[90] H. Sasaki, S. Imamura, and K. Inoue. Coordinated power-performance optimization in many-cores. In *Proceedings of the 22nd International Conference on Parallel Architectures and Compilation Techniques*, pages 51–61, Sep. 2013.

[91] John Shalf, Sudip Dosanjh, and John Morrison. Exascale computing technology challenges. In *Proceedings of the 9th International Conference on High Performance Computing for Computational Science*, VECPAR'10, pages 1–25, Berlin, Heidelberg, 2011. Springer-Verlag.

[92] A. Sharifi, A. K. Mishra, S. Srikantaiah, M. Kandemir, and C. R. Das. Pepon: Performance-aware hierarchical power budgeting for noc based multicores. In *2012 21st International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pages 65–74, Sep. 2012.

[93] N. A. Simakov, R. L. DeLeon, J. P. White, T. R. Furlani, M. Innus, S. M. Gallo, M. D. Jones, A. Patra, B. D. Plessinger, J. Sperhac, T. Yearke, R. Rathsam, and J. T. Palmer. A quantitative analysis of node sharing on hpc clusters using xdmod application kernels.

In *Proceedings of the XSEDE16 Conference on Diversity, Big Data, and Science at Scale*, XSEDE16, pages 32:1–32:8, New York, NY, USA, 2016. ACM.

[94] S. Song, C. Su, B. Rountree, and K. W. Cameron. A simplified and accurate model of power-performance efficiency on emergent gpu architectures. In *2013 IEEE 27th International Symposium on Parallel and Distributed Processing*, pages 673–686, May 2013.

[95] J. Spitaels. Dynamic power variations in data centers and network rooms. *American Power Conversion White Paper*, 5, 2005.

[96] John A Stratton, Christopher Rodrigues, I-Jui Sung, Nady Obeid, Li-Wen Chang, Nasser Anssari, Geng Daniel Liu, and Wen-mei W Hwu. Parboil: A revised benchmark suite for scientific and commercial throughput computing. *Center for Reliable and High-Performance Computing*, 127, 2012.

[97] L. Subramanian, V. Seshadri, A. Ghosh, S. Khan, and O. Mutlu. The application slowdown model: Quantifying and controlling the impact of inter-application interference at shared caches and main memory. In *Proceedings of the 48th International Symposium on Microarchitecture*, MICRO-48, pages 62–75, New York, NY, USA, 2015. ACM.

[98] C. Szegedy, Wei Liu, Yangqing Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, June 2015.

[99] Yefu Wang, Kai Ma, and Xiaorui Wang. Temperature-constrained power control for chip multiprocessors with online model estimation. *SIGARCH Comput. Archit. News*, 37(3):314–324, June 2009.

[100] D. H. Woo and H. S. Lee. Extending amdahl's law for energy-efficient computing in the many-core era. *Computer*, 41(12):24–31, Dec 2008.

[101] Qiang Wu, Qingyuan Deng, Lakshmi Ganesh, Chang-Hong Hsu, Yun Jin, Sanjeev Kumar, Bin Li, Justin Meza, and Yee Jiun Song. Dynamo: facebook's data center-wide power management system. In *Computer Architecture (ISCA), 2016 ACM/IEEE 43rd Annual International Symposium on*, pages 469–480. IEEE, 2016.

[102] Daecheol You. Dynamic voltage and frequency scaling framework for low-power embedded gpus. *Electronics Letters*, 48:1333–1334(1), October 2012.

[103] Gulnara Zhabelova, Alireza Yavarian, and Valeriy Vyatkin. Data center power dynamics within the settings of regional power grid. In *Emerging Technologies & Factory Automation (ETFA), 2015 IEEE 20th Conference on*, pages 1–5. IEEE, 2015.

[104] Huazhe Zhang and Henry Hoffmann. Maximizing Performance Under a Power Cap: A Comparison of Hardware, Software, and Hybrid Techniques. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '16, pages 545–559, New York, NY, USA, 2016. ACM.

[105] Q. Zhu, B. Wu, X. Shen, L. Shen, and Z. Wang. Co-Run Scheduling with Power Cap on Integrated CPU-GPU Systems. In *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 967–977.

[106] P. Zou, T. Allen, C. H. Davis IV, X. Feng, and R. Ge. Clip: Cluster-level intelligent power coordination for power-bounded systems. In *2017 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 541–551, Sept 2017.

[107] P. Zou, X. Feng, and R. Ge. Contention aware workload and resource co-scheduling on power-bounded systems. In *2019 IEEE International Conference on Networking, Architecture and Storage (NAS)*, pages 1–8, Aug 2019.

[108] P. Zou, D. Rodriguez, and R. Ge. Maximizing throughput on power-bounded hpc systems. In *2018IEEE International Conference on Cluster Computing Poster (CLUSTER Poster)*, Sept 2018.

Appendices

A Appendix Experimental Platform

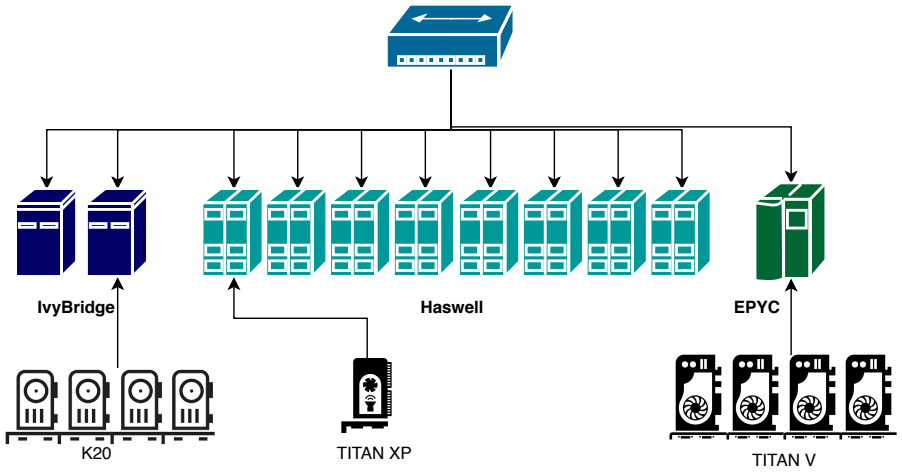


Figure A.1: The architecture of the cluster for evaluation in the proposal.

Table A.1: The parameters of server nodes.

Name	Manufacture	Node count	core type	core count	DRAM	L1 cache	L2 cache	L3 cache	DVFS range
IvyBridge	Intel	2	E5-2670V2	2×10	64 GB DDR3	64 KB + 64 KB	256 KB	25.6 MB	1.2∼2.5 GHz
Haswell	Intel	8	E5-2670V3	2×12	128 GB DDR4	32 KB + 32 KB	256 KB	30 MB	1.2∼2.3 GHz
EPYC	AMD	1	7551P	32	128 GB DDR4	64 KB + 32 KB	512 KB	64 MB	1.2∼2.0 GHz

As shown in Figure A.1, the cluster contains two Intel IvyBridge nodes, 8 Intel Haswell nodes, and 1 AMD EPYC node. Table A.1 lists the parameters of the hosts. Table A.2 shows the detail board configuration of TITAN V. All these nodes are connected by a switch through 10gb network interface. One of the IvyBridge nodes server as login node for the cluster, the other IvyBridge node is the evaluation node for experiment in Chapter 3. The 8 Intel Haswell nodes are used for evaluation in Chapter 3, Chapter 4, Chapter 5 and Chapter 6. The AMD EPYC node will be used for evaluation in Chapter 6 for multiple GPU experiments. For all studies in this proposal, we disable the processor's turbo boost performance state and hyperthreading features.

A.1 Intel IvyBridge

The IvyBridge node has two Intel IvyBridge 10-core E5-2670V2 processors and 64 GB 1600 MHz DDR3 ECC/REG 1.5V memory. Each processor core has a 64 KB L1 instruction cache, a 64 KB L1 data cache, and a unified 256 KB L2 cache. The cores can be scheduled with DVFS among 14 performance states from 1.2 GHz to 2.5 GHz with a step of 0.1 GHz. Each processor package comprises a core domain with the cores and private caches and an uncore domain with a 25.6 MB L3 cache, graphics processors and memory controller.

A.1.1 NVIDIA TESLA K20

One of the IvyBridge node hosts 4 NVIDIA TESLA K20 GPU card. Each Tesla K20 GPU contains 2496 CUDA cores with 5 GB GDDR5 memory. There are 13 streaming multiprocessors (SM) and 192 CUDA cores in a SM. Each multiprocessor supports frequency scaling from 324 MHz to 705 MHz. The GPU enables a power cap starts from 130 watts to the thermal design power limit 230 watts. The K20 GPU has a peak performance at 1.17 TFLOP/S (double precision). Figure A.2 illustrates the architecture of the K20 streaming multiprocessors.

A.2 Intel Haswell

The cluster contains 8 Intel Haswell nodes. Each node has two Intel 12-core Haswell E5-2670 v3 processors and 128GB DDR4 DRAM evenly distributed between two NUMA nodes. Each processor core has 32 KB L1 instruction cache, a 32 KB L1 data cache, and a unified 256 KB L2 cache. All cores support per-core DVFS and 12 frequencies ranging from 1.2GHz to 2.3GHz by 0.1 GHz. Each processor package comprises a core domain with the cores and a core domain with the cores and private caches and an uncore domain with a 30 MB L3 cache, graphics processors and memory controller.

A.2.1 NVIDIA TITAN XP

The first Haswell node hosts a NVIDIA TITAN XP GPU card. The TITAN XP GPU contains 3840 CUDA cores with 12 GB GDDR5X memory. The GPU has 40 streaming multiprocessors, and each SM has 96 CUDA cores. The GPU enables a power cap starts from 130 watts to the thermal design power limit 250 watts. Each multiprocessor supports multiple core frequency scaling and memory frequency scaling. The TITAN XP GPU has a peak performance at 12.1 TFLOP/S (double precision). Figure A.3 illustrates the architecture of the TITAN XP streaming multiprocessors.

A.3 AMD EPYC

The AMD EPYC node has a AMD EPYC 7551P 32-Core processor and 128 GB DDR4 DRAM. The host has 4 NUMA node with each one contains 8 cores. Each processor core has a 64 KB 4-way set associative instruction cache, a 32 KB 8-way set associative writeback data cache, and a 512 KB inclusive 8-way set associative unified cache. The cores can be scheduled with DVFS from 1.2 GHz to 2.0 GHz with a step of 0.1 GHz.

A.3.1 NVIDIA TITAN V

The EPYC node hosts 4 TITAN V GPU card. Each GPU card contains 5120 CUDA cores with 12 GB High Bandwidth Memory (HBM). Each GPU owns 80 streaming multiprocessors (SM) and 640 Tensor Cores. The GPU supports power control starts from 130 watts to the thermal design power limit 250 watts. Figure A.4 illustrates the architecture of the K20 streaming multiprocessors.

Figure A.2: The architecture layout of NVIDIA K20 GPU.

Table A.2: NVIDIA TITAN V, TITAN XP and Tesla K20 configurations [2].

Model	Code Name	CUDA core	Tensor Core	SM Count	Memory	Memory Capacity	Memory Bandwidth	Speed	CUDA computer ability
Titan V	GV100	5120	640	80	HBM2	12GB	652.8 GB/s	61.4 TFLOPS	7.0
Titan XP	GP102	3840	NA	40	GDDR5X	12 GB	547.7 GB/s	11.4 TFLOPS	6.1
Tesla K20	GK110	2496	NA	13	GDDR5	5GB	208 GB/s	1.2 TFLOPS	3.5

B Open Data and Code

The data collected for the dissertation had been uploaded to *Github*. Corresponding scripts and source code are also shared in the repository. The repository address is `https://github.com/pengfei-zou/PowerBound`. The repository is organized by each conference paper that related to this dissertation. Reader can refer to the repository for more experimental details and data collection.

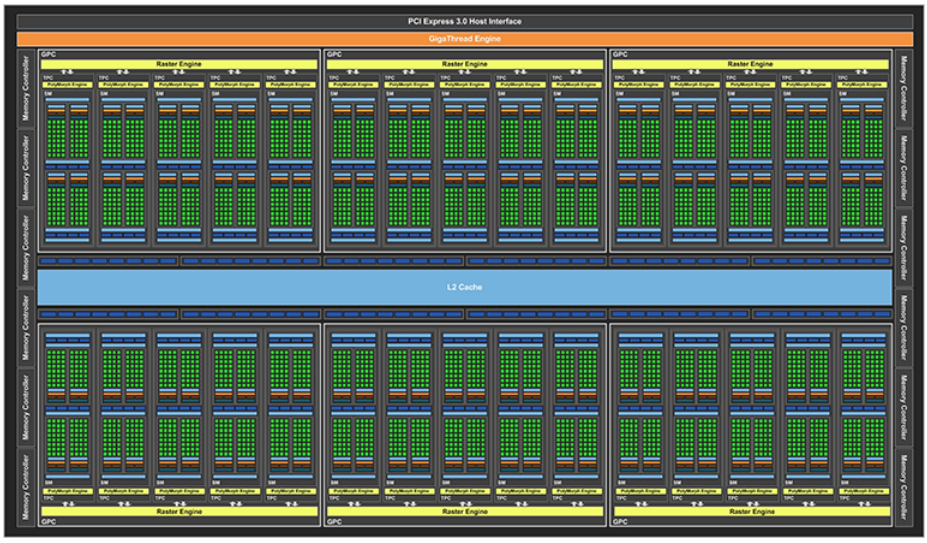


Figure A.3: The architecture layout of NVIDIA TITAN XP GPU.

Figure A.4: The architecture layout of NVIDIA TITAN V GPU.